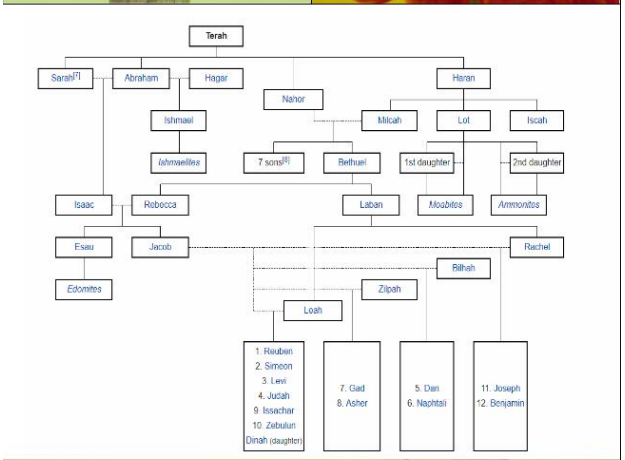
שפות תכנות – תרגיל מספר 6 חלק יבש

תאריך הגשה: *25.1.18*

מגישים:
סהר כהן – 206824088
יובל נהון – 206866832

גילוי עריות:



1.תחימה (scope) – זהו התחום בו binding מסויים קיים – כלומר תהיה משמעות לשם שניתן לישות הזאת לפני כן, ויהיה ניתן לגשת אל הזיכרון במקום של ישות זו לפי שמה.

כריכה (binding) – קישור בין שם לישות. לדוגמא ב-scope של הפונקציה f יש גישה למשתנה בשם x שהוא מיוחס למקום מסויים בזיכרון.

סביבה (environment) – קבוצה של bindings שמהווים אוסף המזהים המוכרים בנקודה מסוימת בתוכנית.

2. העמסה זהו מונח שיש לו פירושים שונים בהתאם להקשר שבו הוא נאמר.

בהקשר של פולימורפיזם, יש העמסת אופרטורים וישויות – קיום מספר ישויות שונות בעלות אותו השם/אופרטור שההבדלה ביניהם נעשית על ידי הקשר (למשל בהעמסת פונקציות נעשית הסקה לגבי איזו פונקציה התכוון המשתמש לפי כמות הפרמטרים וטיפוסם).

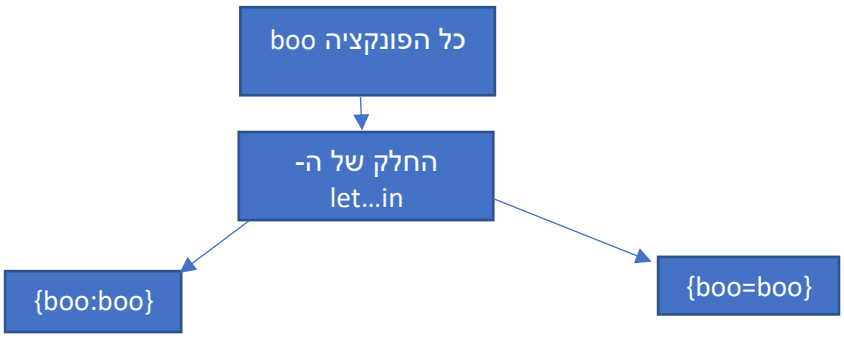
קיים סוג נוסף של העמסה שמתבצע על מילות המפתח של שפה מסוימת. העמסה זו היא שיוך למילת מפתח מספר משמעויות שונות בהתאם להקשר שבה המתכנת השתמש בה.

הסתרה – הסתרה היא מצב שבו הגדרה מחדש של מזהה מסתירה את ההגדרה הקודמת עבור אותו המזהה ולכן יהיה שימוש בהגדרה החדשה.

ההבדל בין העמסה להסתרה הוא שבהסתרה משמעות חדשה מסתירה לגמרי משמעות ישנה, ובהעמסה מספר משמעויות יכולות להתקיים ביחד.

3. בזמן ההגדרה נוצרת סביבה אחת יחידה שהיא הסביבה של הפונקציה boo, וכריכה אחת שהיא הקישור של הפונקציה boo לשמה.

4. בזמן ריצה נוצרות מספר סביבות:



הכריכות שנוצרות הן:

Type boo=boo

Val boo:boo=boo

Val boo:{boo:boo}={boo=boo}

5. (1) הכרזה על שם הפונקציה.

(2) הכרזה על שם של טיפוס חדש בשם boo.

(3) הסתרה של הפרמטר לפונקציה boo שהוגדר ב-(13).

(4) הסתרה של הקבוע בשם boo מ-(3) על ידי קבוע חדש בשם boo מטיפוס record.

(5) הפניה לשדה boo שנמצא בתוך record.

(6) הפניה לטיפוס boo שהוגדר ב-(2).

(7) הסתרה של (4). Boo שהוגדר פה הוא מטיפוס boo.

(8) הפניה ל-record שהוגדר ב-(4).

(9) הפניה לערך הקבוע boo שניתן כפרמטר לפונקציה ב-(13).

(10) הפניה לטיפוס boo שהוגדר ב-(2).

(11) הפניה ל-(7).

(12) הפניה ל-(3).

(13) הכרזה על פרמטר בשם boo.

6. (1) boo

(2) G

(3) a

(4) b

(5) c

(6) G

(7)c

(8)b

(9) k

(10) G

(11) c

(12) k

(13) k

```
fun (1) boo (13)  k =
   let  type (2)  G = int
      val (3)  a : (6)  G = (9)  k
      val (4)  b : { (7)  c : (10)  G } = { (11)  c = (12)  k }
   in
       # (5)  c (8)  b
```

7. הפונקציה מקבלת ארגומנט מטיפוס int ומחזירה אותו.

8. Closure זוהי תכונה של השפה שמאפשרת לגשת לפונקציה ולמשתנים שלה גם לאחר שההקשר של הפונקציה פסק מלהתקיים על ידי שמירת סביבתה ב-heap. כלומר, השפה מאפשרת לשמור את כל המשתנים שמוגדרים ב-scope שלה. על מנת לאפשר קיום של closures על השפה לאפשר קיום של פונקציות מקוננות (כמו שיש ב-ml עם היכולת לכתוב curried functions). לעומת זאת, יכולה להיות שפה שבה יש פונקציות מקוננות אבל לא closure – לדוגמא שפה שבה הפונקציה שומרת את סביבתה אבל לא מאפשרת גישה לסביבתה לאחר סיום חייה.

דוגמא ל-closure בפייתון:

```
def transmit_to_space(message):
    "This is the enclosing function"
    def data_transmitter():
        "The nested function"
        print(message)

    data_transmitter()

print(transmit_to_space("Test message"))
```

בדוגמא זו יש פונקציה מקוננת בשם data_transmitter ששוכנת בתוך הפונקציה transmit_to_space. נשים לב שבריצת התוכנית תודפס ההודעה שנשלחה "Test messege" מהפונקציה המקוננת אך בקריאה לפונקציה הראשית תתבצע הדפסה "None", שכן הפונקציה הראשית לא מדפיסה כלום (רק המקוננת).

9. בשפת קוטלין יש generators – בשפה ניתן ליצור sequences על ידי generateSequence למשל, ויש בה את מילת המפתח היחודית לגנרטורים:,yield,yieldAll.

לדוגמא:

```
val sequence = buildSequence {
    val start = 0
    // yielding a single value
    yield(start)
    // yielding an iterable
    yieldAll(1..5 step 2)
    // yielding an infinite sequence
    yieldAll(generateSequence(8) { it * 3 })
}
```

בשפת קוטלין יש coroutines – ניתן להשתמש ב-coroutines על ידי סימון הפונקציות הרצויות במילה השמורה suspend.

בשפת קוטלין יש גם Continuation שממומש על ידי interface בשם Continuation.