שפות תכנות, 234319

חורף תשע"ח

תרגיל בית מספר 6 - Prolog, ML and Everything In Between

תאריך פרסום: 11.01.2018
מועד אחרון להגשה: 25.01.2018
מועד אחרון להגשה מאוחרת: 28.01.2018
מתרגל אחראי: תומר גלבר
אי-מייל: tomerghelber@cs.technion.ac.il
בפניה בדוא"ל, נושא ההודעה (**subject**) יהיה "**6PLW2017 EX**" (ללא המרכאות).

תרגיל בית זה מורכב משני חלקים, חלק יבש וחלק רטוב. לפני ההגשה, ודאו שההגשה שלכם תואמת את הנחיות ההגשה.

חלק יבש

1. **הסבירו במילותיכן את המונחים <u>סביבה</u> (environment), <u>כריכה</u> (binding), ו<u>תחימה</u> (scoping), ואת הקשר ביניהם.**

<u>Binding</u> – קישור בין שם ליישות.
<u>Environment</u> – סביבה של פונקציה היא קבוצת ה-binding הנגישים לאותה הפונקציה. כלומר, כל המשתנים הנגישים לאותה הפונקציה.
<u>Scoping</u> – המקומות שבהם מוכר ה-binding.

- Static scoping – כאשר פירוש שם המשתנה תלוי במיקום הפונקציה.
- Dymanic scoping – פירוש השם תלוי במצב ריצת התוכנית, כלומר בזמן הריצה.

2. **הסבירות במילותיכן את המונחים <u>העמסה</u> (overloading) ו<u>הסתרה</u> (hiding), ואת ההבדלים ביניהם.**

Hiding – כאשר הגדרה חדשה של מזהה בשפה מסתירה את ההגדרה הקודמת עבור אותו מזהה ולכן יהיה שימוש בהגדרה חדשה. הסתרה נוצרת ב-lexical scoping.
Overloading – מאפשרת להגדיר פונקציה מחדש ב-scope פנימי יותר, כאשר היא כבר הוגדרה ב-scope החיצוני.

ההבדל בין העמסה להסתרה היא שהעמסה נועדה רק להגדיר מחדש פונקציות בעלי טיפוסים שונים לעומת הסתרה שנועדה להגדיר מחדש פונקציה בעלת חתימה זהה לחלוטין(זהה בהגדרה תלוי שפה).

3. **הקוד בהמשך הוא הגדרה חוקית של פונקציה בשפת ML האהובה. הסבירו אילו סביבות (enviroments) ואילו כריכות (bindings) נוצרות בזמן <u>הגדרה</u>(definitions) של הפונקציה boo. תארו או ציירו את ההירכיה של הסביבות (lookup order).**

```
fun boo boo =
  let  type boo = int
     val boo : boo = boo
     val boo : { boo : boo } = { boo = boo }
  in
      # boo boo
  end
```

הסביבה המוגדרת היא רק של הגדרת הפונקצייה boo.
הקשר(binding) היחיד שיש הוא הקישור של המילה boo לפונקציה שהוגדרה.

4. הסבירו אילו סביבות (environments) ואילו כריכות (bindings) נוצרות בזמן <u>הפעלה</u> (application) של הפונקציה boo. תארו או ציירו את ההירכיה של הסביבות (lookup order).

סביבות-

קיימות שלוש סביבות בקוד הנ"ל,
החיצונית ביותר היא הסביבה של הפונקציה בשם boo,
הפנימית יותר היא הסביבה של הlet/in,
והפנימית ביותר היא בתוך ההגדרה של הrecord בL-value(הסוגריים המסולסלים ימני ושמאלי כל אחד בנפרד).

כריכות-

בתוך ה-let קיימת הגדרה(קשירה) של type בשם boo מסוג int.
לאחר מכן יש הגדרה(קשירה) של שני טיפוסים, הראשון בשם boo מסוג boo כך שהוא מחזיק ערך boo.
והשלישי הוא record בשם boo המחזיק איבר אחד בשם boo(השם של השדה).
בתוך ה"השמה" של הrecord יש binding של boo לטיפוס בשם boo.
וגם הקשירה של שם הפונקציה בהתחלה.

ההיררכיה של הסביבות-

ההיררכיה של הסביבות הינה בסדר שמתואר לעיל.
כלומר שלוש היררכיות, מהחיצונית ביותר לפנימית ביותר.

5. נמספר את המופעים של המילה boo לפי סדר הופעתן בקוד עבור על מופע רשמו מה מציין השם boo. ציינו אם מדובר ב<u>הכרזה</u> על שם (declaration) או ב<u>הפניה</u> עליו (reference), וכן אם מדובר ב<u>העמסה</u>(overloading) או ב<u>הסתרה</u>(hiding). הסבירו במידת הצורך מהי הסביבה (environment) ומהי הכריכה (binding) הרלוונטית.

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

(1) הגדרת שם הפונקציה, הכרזה.
(2) הגדרת שם חדש לטיפוס, הכרזה, מוגדר בסביבה של let/in כדי שלא יוכר השם מחוץ לפונקציה.
(3) הגדרת טיפוס מסוג boo בשם boo, הכרזה,הסתרה. בסביבה של let/in.
(4) הגדרת טיפוס מסוג record, הכרזה, הסתרה של משתנה boo שהוכרז שורה לפני.
(5) הפניה שדה boo בrecord, הפניה.
(6) הכרזה על סוג הטיפוס של אותו משנה.
(7) הכרזה על שדה בrecord, הכרזה.
(8) הפניה לערך boo, הפניה.
(9) השמה של ערך פרמטר הפונקציה, הפניה.
(10) הכרזה על סוג הטיפוס בrecord(מסוג boo).
(11) השמה לשדה boo בrecord, הפניה.
(12) השמה של ערך ב(3), הפניה.
(13) פרמטר הפונקציה, הפניה.

6. **החליפו את המופעים של boo בשמות חדשים כלשהם באופן עקבי כך שלא תהיה העמסה או הסתרה של שמות אך תוך שימור משמעות הקוד. יש לרשום את השמות במסגרות הממוספרות.**

```
fun (1) _f__ (13) __a_ =
  let type (2) _num__ = int
    val (3) _y__ : (6) __num_ = (9) __a_
    val (4) _x__ : { (7) _x1__ : (10) __num_ } = { (11) _x1__ = (12) __y_ }
  in
    # (5) _x1__ (8) __x_
  end;
```

7. **מה עושה boo?**

כלום, מחזיר את הערך אותו הוא קיבל.

8. **מה זה closure, ואיך המושג קשור לפונקציות מקוננות? בפרט, האם יכולה להיות שפה עם פונקציות מקוננות ובלי closure, או בלי פונקציות מקוננות ועם closure? נמקו והביאו דוגמה מבוארת לשימוש ב-closure בשפת Python.**

Closure – הן פונקציות אשר גם לאחר שההקשר שלהן לא תקף עוד, הן עדיין שומרות על הסביבה הפנימית שלהן.

פונקציה מקוננת שומרת את סביבתה. אך היא לא בהכרח מאפשרת גישה לסביבתה לאחר סיום חייה.

פונקציה שמקיימת Closure היא סוג של פונקציה מקוננת השומרת את סביבתה בערימה, כך שמתאפשרת גישה לסביבתה גם לאחר סיום ריצתה.

משל פסקל, יש בה פונקציות מקוננות שאינן מקיימות Closure.

Closure הוא מקרה פרטי של פונקציה מקוננת לכן מקרה הפוך הינו יכול להיות.

דוגמא לClosure בPython

```
>>> def counter_factory():
...     count = 0 # here I create the variable to increment
...     def counter():
...         nonlocal count
...         count += 1
...         return count
...     return counter
...
```

9. אילו הפשטות מתקדמות (כפי שנלמדו בפרקים 7.2 ו-7.4 בחוברת ההרצאות) קיימות בשפת Kotlin?

בקוטלין קיימות ההפשטות הבאות:

- dynamic scoping
- nested functions(Thunks)
- Lexical scoping
- generators (מחלקה חיצונית)
- Coroutines
- Closures