~~0. מעבר על הקוד הקיים~~

~~1. תשתית (שקפים 112-114, פרק 6) - "לתמוך בכלום בצורה מושלמת"~~

~~a. כל הקבצים מהעבודות מתחברים זה לזה ומדפיסים שגיאה not\_implement~~

~~b. קוראים את הsrc file ומשרשרים אליו את הקבצי ספריה שלנו בscheme (וכמובן מוסיפים את קוד הscheme שקיבלנו (stdlib.scm))~~

~~c. כותבים קוד לבניית טבלת קבועים (שקף 36)~~

~~d. כותבים קוד לבניית טבלת משתנים חופשיים (שקף 63)~~

~~e. בניית השלד (Prologue + Epilogue + nasm + gcc...)~~

~~e. כותבים סקריפט טסט לבדיקה כמו שהם יעשו (סעיף 5) – מה שהבחור שלח בוואטסאפ~~

**(סוף יום 1)**

~~2. Consts & test~~

~~3. Seq & test~~

~~4. If & test~~

~~5. And & test~~

~~6. Or & test~~

~~7. Define & set - free variables & test~~

**(יום 3)**

8. ~~מבטלים את ApplicTP'~~ ו:

~~a. LambdaSimple~~

* ~~calc |Env| - not necessarily in assem~~
* ~~allocate ExtEnv – size of 1 + |Env|~~
* ~~copy pointers of minor vectors from Env (through stack) to ExtEnv with offset of 1~~
* ~~Get the number of params from stack~~
* ~~Allocate ExtEnv[0] to point to a vector in the size of Params (as stored in the ArgsNum on stack)~~
* ~~copy parameters from the stack to that vector~~
* ~~generate the Lcode label~~

~~b. Parameters / get~~

~~c. Paramenets / set~~

~~d. Bound vars / get~~

~~e. Bound vars / set~~

~~f. Box / get~~

~~g. Box / set~~

**(יום 5)**

~~8ב. Applic~~

~~Test a lot - כאן יהיו רוב הבאגים, צריך להגיע לזה מהר ולהקדיש לזה זמן~~

~~למבדות~~

rec1 – הפונקציה הרקורסיבית func דורשת בוקס למיטב הבנתי.

בזמן יצירת הclosure של ה lambda הפנימית (זו שנמצאת בrhs של הset! אחרי שמרחיבים את הLetrec לlambda simple, הקוד שאנחנו מייצרים נמצא בהערה בתוך הטסט). אנחנו מרחיבים את הסביבה.

בזמן ההרחבה של הסביבה אנחנו מעתיקים לסביבה את PVAR(0) לheap למקום ה[0][0] שכרגע הוא מצביע לסימבול whatever בטבלת הקבועים. אחרי שאנחנו מסיימים לייצר את הclosure (שסוחב איתו את הסביבה) אנחנו דורסים עם set! את המשתנה PVAR(0) על הstack עם הכתובת לclosure שהרגע ייצרנו. בסביבה של הclosure בheap עדיין יש מצביע לסימבול של whatever (כפי שהיה בstack). לכן כשאנחנו באים להפעיל את func בתוך הif, אנחנו פונים ל VarBound(func,0,0) שמכיל מצביע לסימבול whatever. כשאנחנו מנסים לקרוא לזה, יש segfault (אם משאירים את הבדיקה של הclosure type אז יקפוץ int3) .

אם עושים בוקס (הכרחתי בsemantic-analyser לעשות בוקס להכל) אז הקוד עובד נפלא.

[צריך לשים לב שאם עושים include ל stdlib.scm אז יש segfault בלי שום קשר כי גם שם יש פונקציות רקורסיביות]

**(יום 6)**

9. ניתן לבדוק את (תלוי רק במשתנים חופשיים ואפליקציות):

+ (binary version), -> test arithmetic2 does some crazy shit, but I’m not using stdlib.scm so that’s probably why (?)

\* (binary version), -> test arithmetic3, same shit

/ (binary version), -> test arithmetic4, same shit

< (binary version), -> test arithmetic5, same shit

= (binary version), -> test arithmetic6, same shit

(we need to check the thing with fvar+15 and fvar+8\*15)

boolean?,

char->integer,

char?,

denominator,

eq?,

exact->inexact,

flonum?,

gcd (binary version),

integer->char, make-string (binary version), null?, pair?, procedure?, rational?, string-length, string-ref, string-set!, string?, symbol?, symbol->string.

9ב. ניתן לבדוק את (תלוי ב lambda simple):

a. not

**(יום 7)**

9ג. מתחילים לממש פרימיטיבים

a. מממשים באסמבלי (סעיף 4.1)

~~car, cdr, cons, set-car!, set-cdr!~~

**ובודקים**

b. מממשים בscheme את:

~~fold-left (non-variadic!), fold-right (non-variadic!), and cons\* (variadic)~~

**ובודקים**

**(יום 8)**

9ד. ניתן לבדוק את:

integer?, number?, zero?

9ה. ניתן לבדוק את:

a. \* (variadic), + (variadic), - (variadic), / (variadic), < (variadic), = (variadic), > (variadic), append (variadic), equal?, gcd (variadic), length, make-string, map (variadic), string->list.

9ו. לא לשכוח לבדוק pset

**(יום 9)**

10. מחזירים את ApplicTP'

a. מוסיפים תמיכה בTP

b. Test a lot - גם כאן יהיו הרבה באגים

c. מימוש באסמבלי של apply (variadic)

**(יום 11)**

11. LambdaOpt & test

12. מוסיפים את שאר הפרימיטיביים (תלויים בlambda opt)

List

**(יום 13)**