```
#!/usr/bin/env/stack
-- stack --install-ghc runghc

-- Homework 2

-- Authors: Jack Neff, Rhea Mae Edwards
-- Date: 05/03/2017

main :: IO()
main = putStrLn("Program Executed.")

-- Exercise 1: A Stack Language

-- Abstract Syntax for Cmd
data Cmd = LD Int
    | ADD
    | MULT
    | DUP
    deriving Show

-- Stack as a List of Integers
type Stack = [Int]

type Prog = [Cmd]

type D = Maybe Stack -> Maybe Stack

-- Semantic Domain as Function Domain
sem :: Prog -> D
sem [] a = a
sem (x:xs) a = sem xs (semCmd x a)

-- Auxiliary Function for Individual Operations
semCmd :: Cmd -> D

semCmd (LD a) xs = case xs of Just xs -> Just ([a] ++ xs)
                              _       -> Nothing

-- Error Domain
eval :: Prog -> Maybe Stack
eval p = sem p (Just [])


{-- This exercise is the victim of a persisten type error and wont compile

-- Exercise 2: Extending the Stack Language by Macros

-- (a) Abstract Syntax
-- Represents macro definition and calls
-- Gives a correspondingly changed data definition for Cmd

data Cmd2 = LD2 Int | ADD2 | MULT2 | DUP2 | DEF String Prog2 | CALL String
```

```
-- (b) New Type State
-- Which includes macro definitions and the stack
-- Multiple macro definitions in a list
-- Each macro definition represented by a pair,
-- first component a macro name, second component the sequence of commands

-- New Type Macro Definition
type Macros = [(String, Prog2)]

type Prog2 = [Cmd2]

-- New Type State
type State = (Macros, Stack)

type D2 = State -> Maybe State

-- (c) Semantics for Extended Language as Function sem2

sem2 :: Prog2 -> D2
sem2 [] (m, s) = Just (m, s)
sem2 (x:xs) (m, s) = sem2 xs (m, s) (semCmd2 x (m, s)) -- TYPE ERROR HERE

semCmd2 :: Cmd2 -> D2
semCmd2 (LD2 i) (m, s) = Just (m, [i] ++ s)
semCmd2 (ADD2) (m, s) = case s of (s1:s2:s) -> Just (m, [s1+s2] ++ s)
                                  _         -> Nothing
semCmd2 (MULT2) (m, s) = case s of (s1:s2:s) -> Just (m, [s1*s2] ++ s)
                                   _         -> Nothing
semCmd2 (DUP2) (m, s) = case s of (s1:s) -> Just (m, [s1, s1] ++ s)
                                  _      -> Nothing
semCmd2 (DEF str p2) (m, s) = Just ([(str, p2)] ++ m, s)
semCmd2 (CALL str) (m, s) = sem2 ((semCmd3 m str)) (m, s)

semCmd3 :: Macros -> String -> Prog2
semCmd3 ([(s, p)]) str = case s of s == str ->  p    -- Macro name matches stored value
                       s != str -> Nothing -- Not the right macro

--}

-- Exercise 3: Mini Logo

-- Abstract Syntax
data Cmd3 = Pen Mode
          | MoveTo Int Int
          | Seq Cmd3 Cmd3
          deriving Show

data Mode = Up | Down
          deriving (Show, Eq)

-- Semantics of Mini Logo, Set of Drawn Lines
-- Maintained by keeping track of current position and status
type State = (Mode,Int,Int)
```

```
-- Semantic Domain Set of Drawn Lines
type Line = (Int,Int,Int,Int)
type Lines = [Line]

-- For each Cmd modifications of the current drawing state
-- And what lines it produces
semS :: Cmd3 -> State -> (State,Lines)
semS (Pen m1)       s@(m2, x, y) | m1 /= m2              = ((m1, x, y), [])
                                 | otherwise             = (s, [])
semS (MoveTo x1 y1) (m, x2, y2)  | m == Up               = (ns, [])
                                 | x1 /= x2 && y1 /= y2  = (ns, [(x2, y2, x1, y1)])
                                 | otherwise             = (ns, [])
                                 where ns = (m, x1, y1)
semS (Seq a b) s = (fst s2, snd s1 ++ snd s2)
                 where
                    s1 = semS a s
                    s2 = semS b (fst s1)

-- Function calls semS
-- Initial state, pen up and current drawing position at (0,0)
sbegin = (Up, 0, 0)

sem' :: Cmd3 -> Lines
sem' a = snd (semS a sbegin)
```