module Stack where

-- This is based on the last part of Assignment 1.

-- In this version, we’ll use a *functional* context. That is, a context
-- will be a function mapping strings to ints.
type Ctxt = String -> Int

-- An empty context maps all strings to error
emptyCtxt :: Ctxt
emptyCtxt v = error (v ++ " not found")

extendCtxt :: Ctxt -> String -> Int -> Ctxt
extendCtxt ctxt new_var new_val
  = \query -> if query == new_var then new_val else ctxt query
  -- This creates a new function that checks if the query matches the new
  -- binding. If so, return the new value. Otherwise, look it up in the
  -- original context.

buildContext :: [(String, Int)] -> Ctxt
buildContext []                  = emptyCtxt
buildContext ((var, val) : rest) = extendCtxt (buildContext rest) var val

-- One instruction in our stack machine
data Insn
  = IPushC Int        -- push an int64 constant onto the stack
  | IPushV String     -- push (lookup string ctxt) onto the stack
  | IMul              -- multiply the top two values on the stack
  | IAdd              -- add the top two values on the stack
  | INeg              -- negate the top value on the stack
deriving (Eq, Show)

-- A stack program is just a list of instructions.
type Program = [Insn]

-- The stack; the head of the list is the top of the stack.
type Stack = [Int]

-- Carry out one instruction, in a given context, producing an output stack.
step :: Ctxt -> Stack -> Insn -> Stack
step _ s             (IPushC n) = n : s
step c s             (IPushV x) = c x : s
step _ (v1 : v2 : s) IMul       = v1 * v2 : s
step _ (v1 : v2 : s) IAdd       = v1 + v2 : s
step _ (v : s)       INeg       = (-v) : s
step _ _ _            = error "stack had too few values"

-- Executing a program means repeatedly processing instructions
execute :: Ctxt -> Stack -> Program -> Stack
execute _ s []         = s -- no more instructions to execute
execute c s (i : cont) = execute c (step c s i) cont

-- Extract the final, sole value from the stack. It must have 1 element.
answer :: Stack -> Int
answer [n] = n
answer _    = error "no answer"

-- Run a program in a given context for its variables
run :: Ctxt -> Program -> Int
run c p = answer (execute c [] p)
Stack.hs

73: p1 = [IPushC 2, IPushC 3, IMul]
74: answer1 = run emptyCtxt p1