Stack.hs

1: {- Author: Richard Eisenberg, inspired by Steve Zdancewic
2: File: Stack.hs
3: }
4: Demonstrates ways of working with a stack.
5: -}
6: module Stack where
7:
8: {-# OPTIONS_GHC -W -Wno-unused-imports #-}
9: module where
10: -- This is based on the last part of Assignment 1.
11: -- In this version, we’ll use a *functional* context. That is, a context
12: -- will be a function mapping strings to ints.
13: type Ctxt = String -> Int
14: 15: emptyCtxt :: Ctxt
16: 17: extendCtxt :: Ctxt -> String -> Int -> Ctxt
18: 19: buildContext :: [(String, Int)] -> Ctxt
20: 21: -- An empty context maps all strings to error
22: emptyCtxt v = error (v ++ " not found")
23: extendCtxt ctxt new_var new_val
24:   = \query -> if query == new_var then new_val else ctxt query
25:   -- This creates a new function that checks if the query matches the new
26:   -- binding. If so, return the new value. Otherwise, look it up in the
27:   -- original context.
28: buildContext :: [(String, Int)] -> Ctxt
29: buildContext [] = emptyCtxt
30: buildContext ((var, val) : rest) = extendCtxt (buildContext rest) var val
31: 32: -- One instruction in our stack machine
33: data Insn
34:   = IPushC Int        -- push an int64 constant onto the stack
35:   | IPushV String     -- push (lookup string ctxt) onto the stack
36:   | IMul              -- multiply the top two values on the stack
37:   | IAdd              -- add the top two values on the stack
38:   | INeg              -- negate the top value on the stack
39:   deriving (Eq, Show)
40: 41: -- A stack program is just a list of instructions.
42: type Program = [Insn]
43: 44: -- The stack; the head of the list is the top of the stack.
45: type Stack = [Int]
46: 47: step :: Ctxt -> Stack -> Insn -> Stack
48: step _ s (IPushC n) = n : s
49: step c s (IPushV x) = c x : s
50: step _ (v1 : v2 : s) IMul       = v1 * v2 : s
51: step _ (v1 : v2 : s) IAdd       = v1 + v2 : s
52: step _ (v : s) INeg             = (-v) : s
53: step _ _ _ = error "stack had too few values"
54: 55: execute :: Ctxt -> Stack -> Program -> Stack
56: execute _ s []         = s  -- no more instructions to execute
57: execute c s (i : cont) = execute c (step c s i) cont
58: 59: answer :: Stack -> Int
60: answer n = n
61: answer _ = error "no answer"
62: 63: run :: Ctxt -> Program -> Int
64: run c p = answer (execute c [] p)
65: 66: -- Example:
Stack.hs

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