-- 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

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

-- A stack program is just a list of instructions.

data Program = [Insn]

-- A stack machine has a data space for the stack, as well as the index of one
-- past the top of the stack (the bottom is always 0), the instructions,
-- and the program counter (PC), which tells us which instruction to run
-- next.

data Machine = M { stack        :: IOVector Int
                  , spRef        :: IORef Int
                  , instructions :: Vector Insn
                  , pcRef        :: IORef Int }

newMachine :: Int -> Program -> IO Machine
newMachine size prog = do
  st <- M.new size
  sp <- newIORef 0   -- SP starts at 0
  let insns = I.fromList prog
  pc <- newIORef 0   -- PC starts at 0
  pure (M { stack = st

StackIO.hs

73:           , spRef        = sp
74:           , instructions = insns
75:           , pcRef        = pc }
76: }
77:
78: -- Print to stdout the current machine state (without the context)
79: printMachine :: Machine -> IO ()
80: printMachine (M { stack        = st
81:               , spRef        = sp
82:               , instructions = insns
83:               , pcRef        = pc }) = do
84:   putStrLn "Stack:"
85:   sp_val <- readIORef sp
86:   putStrLn "  SP --> ", putStrLn "  SP --> 
87:   forM_ (reverse [0..sp_val-1]) $ \ stack_loc -> do
88:     stack_val <- M.read st stack_loc
89:     putStrLn ("         " ++ show stack_val)
90:   putStrLn "Instructions:"
91:   pc_val <- readIORef pc
92:   forM_ [0 .. I.length insns - 1] $ \ insn_loc -> do
93:     if (insn_loc == pc_val)
94:       then putStr "  PC --> 
95:       else putStr "         
96:     let insn = insns I.! insn_loc
97:     putStrLn (show insn)
98:   putStrLn " -- Run the machine by one step in the given context."
99:   -- Returns whether the machine is done running (True means "done")
100: step :: Ctxt -> Machine -> IO Bool
101: step ctxt (M { stack        = st
102:             , spRef        = sp
103:             , instructions = insns
104:             , pcRef        = pc }) = do
105:   -- Fetch the instruction
106:   pc_val <- readIORef pc
107:   let insn = insns I.! pc_val
108:   -- Perform the instruction
109:   case insn of
110:     IPushC n -> push n
111:     IPushV x -> push (ctxt x)
112:     IMul     -> do n1 <- pop
113:       n2 <- pop
114:       push (n1 * n2)
115:     IAdd     -> do n1 <- pop
116:       n2 <- pop
117:       push (n1 + n2)
118:     INeg     -> do n <- pop
119:       push (-n)
120:     -- Increment the PC
121:     let new_pc_val = pc_val + 1
122:     writeIORef pc new pc_val
123:   putStrLn (" -- Increment the PC")
124:   let new_pc_val = pc_val + 1
125:   writeIORef pc new pc_val
126:   pure (new pc_val == I.length insns)
127:   where
128:     push n = do
129:       sp_val <- readIORef sp
130:       when (sp_val == M.length st) $ 
131:         error "Out of stack space"
132:     pop = do
133:       sp_val <- readIORef sp
134:       when (sp_val == 0) $ 
135:         error "stack underflow"
StackIO.hs

145:   let new_sp_val = sp_val - 1
146:   writeIORef sp new_sp_val
147:   M.read st new_sp_val
148:
149: -- Executing a machine means repeatedly processing instructions
150: execute :: Ctxt -> Machine -> IO ()
151: execute ctxt m = do
152:   done <- step ctxt m
153:   when (not done) $
154:   execute ctxt m
155:
156: -- Extract the final, sole value from the machine. The stack must have 1 element.
157: answer :: Machine -> IO Int
158: answer (M { stack = st
159:   , spRef = sp }) = do
160:   sp_val <- readIORef sp
161:   when (sp_val /= 1) $
162:   error ("Stack has " ++ show sp_val ++ " values at end of run.")
163:
164: M.read st 0
165:
166: -- Run a program in a given context for its variables, with a given stack size
167: run :: Int -> Ctxt -> Program -> IO Int
168: run size ctxt prog = do
169:   m <- newMachine size prog
170:   execute ctxt m
171:   answer m
172:
173: -- Example:
174: p1       = [IPushC 2, IPushC 3, IMul]
175: answer1  = run 10 emptyCtxt p1
176: overflow = run 1 emptyCtxt p1
177: