Pamphlet 2, INF222, Spring 2021

2.1 Calculator with registers

The following shows the abstract syntax for a register based calculator on integers. Note how `CalcExprAST` has been extended with a case `Reg Register`, where `Register` is an enumeration of 10 distinct register names.

```haskell
-- | AST for register based integer calculator.
--
-- Author Magne Haveraaen
-- Since 2020-03-14

module CalculatorRegisterAST where


-----------------------

-- | Expressions for a calculator with 10 registers.
-- The calculator supports literals and operations
-- Addition, multiplication, and subtraction/negation.
data CalcExprAST
  = Lit Integer
  | Add CalcExprAST CalcExprAST
  | Mult CalcExprAST CalcExprAST
  | Sub CalcExprAST CalcExprAST
  | Neg CalcExprAST
  | Reg Register
  deriving (Eq, Read, Show)

-- | Statement for setting a register
data CalcStmtAST
  = SetReg Register CalcExprAST
  deriving (Eq, Read, Show)

-- | Enumeration of the 10 registers.
data Register
  = Reg0
  | Reg1
  | Reg2
  | Reg3
  | Reg4
  | Reg5
  | Reg6
  | Reg7
  | Reg8
  | Reg9
  deriving (Eq, Read, Show)


-----------------------

-- | A few ASTs for register based CalcExprAST.
```

```
calculatorRegisterAST1
  = Lit 4
calculatorRegisterAST2
  = Neg (Mult (Add (Lit 3) (Sub (Lit 7) (Lit 13))) (Lit 19))
calculatorRegisterAST3
  = Add (Reg Reg1) (Reg Reg4)
calculatorRegisterAST4
  = Reg Reg2

-- | A few ASTs for setting registers CalcStmtAST.
calculatorSetRegisterAST1
  = SetReg Reg4 calculatorRegisterAST1
calculatorSetRegisterAST2
  = SetReg Reg1 calculatorRegisterAST2
calculatorSetRegisterAST3
  = SetReg Reg2 calculatorRegisterAST3
calculatorSetRegisterAST4
  = SetReg Reg1 calculatorRegisterAST4

-----------------
```

The use of registers also introduces statements `CalcStmtAST` for setting values into registers.

The AST file ends with some example expressions and statements in the register calculator language.

2.2 Store

The introduction of registers induces the need for a store to keep track of the register values.

```
-- | Semantics for register based integer calculator.
-- The values of the registers are stored in a Store.
--
-- Author Magne Haveraaen
-- Since 2020-03-14

module CalculatorRegisterStore where

-- | Use Haskell's array data structure
import Data.Array


-----------------

-- | A Store for a register calculator is an array with 10 integer elements.
-- The access functions getregister/setregister need to translate between register and array index.
type Store = Array Integer Integer

-- | Defines a store for 10 registers
 registerstore :: Store
 registerstore = array (0,9) [(i,0) | i <-[0..9]]

-- | Get the value stored for the given register.
getstore :: Store -> Integer -> Integer
getstore store ind =
  if 0 <= ind && ind < 10
  then store ! ind
  else error $ "Not␣a␣register␣index␣" ++ (show ind)

-- | Set the value stored for the given register.
setstore :: Integer -> Integer -> Store -> Store
setstore ind val store =
  if 0 <= ind && ind < 10
  then store // [(ind, val)]
  else error $ "Not␣a␣register␣index␣" ++ (show ind) ++ "␣for␣" ++ (show val)
```

The store above handles 10 distinct indices and stores integers. The store is initialised to contain only zeroes in `registerstore`. It also explicitly checks, in the functions `getstore` and `setstore`, that only integers `0..9` are used as indices.

The store is implemented using the Haskell standard library `Array` data structure, see chapter 14 of `https://www.haskell.org/onlinereport/haskell2010/` for more details.

2.3 Task

The task is again to implement an interpreter, this time for the register calculator abstract syntax. The interpreter needs three functions:

- `evaluate :: CalcExprAST -> Store -> Integer`
  to evaluate a calculator expression given a store.

- `execute :: CalcStmtAST -> Store -> Store`
  to set the value of a calculator expression to a register in the store.

- `getregisterindex :: Register -> Integer`
  to map a register to an index in the store.