Pamphlet 1, INF222, Spring 2021

1.1 Abstract Syntax

This first series of pamphlets goes from implementing simple calculators to a full fledged programming language.

Defining a programming language consists of defining its syntax and semantics. Studying programming languages is about understanding design choices for programming languages, but also some of the tools being used in the domain of programming languages.

For understanding a programming language the syntax is given as *abstract syntax*. This can be given in one of the abstract syntax languages of Ralf Lämmel's Software Languages book: Basic Signature Language (BSL) or Extended Signature Language (ESL). In these pamphlets we will use Haskell code, which has a fairly direct interpretation as BSL or ESL.

1.2 Simple Calculator

The following shows the abstract syntax for expression in a simple calculator on integers.

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

module CalculatorAST where


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

-- | Expressions for a simple calculator.
-- 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
  deriving (Eq, Read, Show)


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

-- | A couple ASTs for CalcExprAST.
calculatorAST1 = Lit 4
calculatorAST2
  = Neg (Mult (Add (Lit 3) (Sub (Lit 7) (Lit 13))) (Lit 19))

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

We assume you can guess the intended semantics from the names of the alternatives of `CalcExprAST`.

1.3 Interpreters

In our course we use *interpreters* to define the semantics of ASTs. They are straight forward to implement, since they for each construct of the AST define the semantics of that construct.

In the programming language community our form of interpreters are often known as *big step operational semantics*. They are also called *abstract machines* and as such are used to define the semantics of programming languages like C and C++.

Interpreters for expressions *evaluate* the expression while those for statements *execute* the statement.

1.4 Task

Implement an interpreter for the simple calculator abstract syntax `CalcExprAST` (expressions). Every such expression can be evaluated directly to a Haskell integer.

Our simple calculator can be treated as a simplified version of the book's Basic TAPL Language (BTL).