Pamphlet 1, INF222, Spring 2023

1.1 Abstract Syntax

This first series of pamphlets invetigates various notions of calculators. These will range from a simple value oriented calculator, via register calculators and calculators with variables, to a framework for calculators for arbitrary domains.

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 we are interested in its *abstract syntax*. This can be given in one of the abstract syntax languages of Ralf Lämmel's Software Languages book (SLBook): Basic Signature Language (BSL) or Extended Signature Language (ESL), or an industry standard like BNF/EBNF[1] or ASDL[2]. In these pamphlets we will use Haskell code, which has a fairly direct interpretation as either of these syntax formalisms.

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 Pam1AST 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))

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

[1] Variants BNF/EBNF are used in the syntax definition of the Ada, Fortran standards.

[2] `https://en.wikipedia.org/wiki/Abstract-Type_and_Scheme-Definition_Language` – ADSL is being used for the abstract syntax in the LCompiler suite for Python and Fortran.

We assume you can guess the intended semantics from the names of the cases in `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 research community our form of interpreters are often known as *big step operational semantics*. They are also called *abstract machines*. Abstract machines are used to define the semantics of programming languages like C, C++, Fortran and Java. Such abstract machines are typically described by English text.

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 SLBook's Basic TAPL Language (BTL).