1. Here is how you can define a tree structure using recursive types in Haskell:

```haskell
data Tree a = Nil | Node a (Tree a) (Tree a) deriving (Show)
```

Write a function `dfs` and a function `bfs` that traverse a given tree in depth-first, respectively breadth-first, order and output the nodes in the respective order.

For example, for the tree:

```haskell
sampleTree = Node "A"
          (Node "B"
           (Node "C" Nil Nil)
           (Node "D" Nil Nil))
          (Node "E"
           (Node "F" Nil Nil)
           (Node "G" Nil
            (Node "H"
             (Node "I" Nil Nil)
             Nil))
          )
```

`dfs sampleTree` should return `"A", "B", "C", "D", "E", "F", "G", "H", "I"`,

2. Define a user-defined type for operator trees. An operator tree contains operands of type integer that are connected via the binary operation addition (+). The smallest possible operator tree is one that only contains one operand. The following diagram shows an example:
Write a function `evaluate` that gets an integer operator tree and evaluates it, i.e., it traverses the tree and adds up all the operands.

3. Now rewrite your user-defined type from the previous exercise to make it an operator tree that contains operands of any type `a` and any binary operation `a→a→a` defined on type `a`. Again write a function `evaluate` that gets an operator tree and evaluates it.

4. Write a Haskell program that reads a list of numbers (entered by a user). The input is terminated by entering a 0. The program should then compute the sum and the product of the list of numbers. Try to compile your program as a stand-alone executable.

Hints:

- Use the function `read` to convert an input into a number (e.g. an `Int`). The function is used like this:

  ```haskell
  number <- getLine
  let intNumber = (read number)::Int
  ```

- To convert a number back into a string (for output), you can use the function `show`
5. Change your implementations of the sieve of Eratosthenes (exercise sheet 7) so that you can compile your programs as stand-alone executable. The program must get the required argument from the command line:

> ./sieve 10
[2,3,5,7]