Exercise Set 5 — 22 October

5.1 Files

Write standalone Haskell programs that work like the Unix `cat`, `tac`, `rev`, and `sort` commands (with no comamand-line options).

5.2 Booleans and numbers

This exercise concerns the language of booleans and natural numbers from Chapter 3 of our textbook, *Types and Programming Languages* by Benjamin Pierce.

In Haskell, we will adopt this data type to represent the terms of the language:

```
data Term  =  Tru
           |  Fls
           |  If Term Term Term
           |  Zero
           |  Succ Term
           |  Pred Term
           |  IsZero Term
           deriving Eq

instance Show Term where
```

(Complete the `Show` instance so terms will be printed to match the concrete syntax.)

We will adopt the following fully bracketted concrete syntax:

```
Term ::= true
       | false
       | if Term then Term else Term fi
       | 0
       | succ ( Term )
       | pred ( Term )
       | iszero ( Term )
```

in which the terminal symbols are the keywords `true`, `false`, `if`, `then`, `else`, `fi`, `succ`, `pred`, `iszero`, the numeral `0`, and the punctuation characters `(` and `)`, represented in Haskell as:

```
data Token = TokenTrue
           | TokenFalse
           | TokenIf
           | TokenThen
           | TokenElse
           | TokenFi
           | TokenSucc
```

```
  | TokenPred
  | TokenIsZero
  | TokenZero
  | TokenLPar
  | TokenRPar
```

(1) Write a scanner function `scan :: String -> Maybe [Token]`. White space is not required around parentheses.

(2) Write a parser function `parse :: [Token] -> Maybe Expr`.

(3) Write a `main` function that reads in a source file in the language of booleans and natural numbers, scans it, parses it, and prints out an abstract syntax tree (an `Expr`) for syntactically valid inputs, or otherwise prints a suitable error message.