Assignment 6
Advanced functional Programming
Topic: Parsing – Lexical and Syntactical Analysis
Issued on: 05/18/2010, due date: 06/01/2010

For this assignment a Haskell script named `AssFFP6.hs` shall be written offering functions which solve the problems described below. This file `AssFFP6.hs` shall be stored in the home directory of your individual account (not of your group account), as usual on the top most level. Comment your programs meaningfully. Use constants and auxiliary functions, where appropriate.

Consider again the programming language **While**, which has been introduced previously:

```
Prog       ::= begin Stmt end
Stmt       ::= AssStmt | IfStmt | RepeatStmt | CompStmt | Skip
AssStmt    ::= Idf := AExpr
IfStmt     ::= if Bexpr then Stmt else Stmt fi
WhileStmt  ::= while Bexpr do Stmt od
Skip       ::= skip
CompStmt   ::= (Stmt ; Stmt)
```

For convenience, we also recall the grammar generating the set of arithmetic and Boolean expressions.

```
Expr   ::= AExpr | Bexpr

AExpr  ::= Term | AExpr Aop Term
Term   ::= Factor | Term Mop Factor
Factor ::= Opd | (AExpr)
Opd    ::= Numeral | Idf
Aop    ::= + | -
Mop    ::= * | /

Bexpr  ::= (Aexpr Relop Aexpr)
Relop  ::= = | /= | > | <
```

We recall that `Idf` denotes an arbitrary identifier and that each identifier is a non-empty sequence of lower case and upper case letters and digits starting with a letter. Moreover, we recall that `Numeral` denotes an unsigned decimal number (i.e., a natural number).

- Extend your

  1. combinator parser *pc* and
  2. monadic parser *pm*

  to parsers

  1. combinator parser *xpc* and
  2. monadic parser *xpm*

  which behave as their counterparts *pc* and *pm*, but which return for identifiers and numerals in addition the identifier and the numeral read. To this end we modify the data type `Token` to a new data type `XToken` as follows:

```
data XToken = Prog |
              Id String | AssOp | Num Integer |
              LeftParenth | RightParenth |
              Plus | Minus | Mult | Div |
              Equal | Unequal | Greater | Less |
              BeginSymb | EndSymb |
              IfSymb | ThenSymb | ElseSymb | FiSymb |
              WhileSymb | DoSymb | OdSymb |
              SkipSymb |
              SemicolonSymb |
              Err
              deriving Show
```

  Take care to implement in addition to required auxiliary functions two functions `main_xpc :: String -> [Token]` and `main_xpm :: String -> [Token]` allowing to test the functioning of your parsers. The token `Err` shall be used by both parsers, if the input string contains a substring, which does not correspond to one of the tokens above. The remainder of the input string shall then be discarded; `err` is then the last token in the result list of the functions `main_xpc` and `main_xpm`.