CMSC 22610 Winter 2015	Implementation of Computer Languages I	Project 1 January 9, 2015

Flang lexer

Due: January 23, 2015

1 Introduction

Your first assignment is to implement a lexer (or scanner) for **Flang**, which will convert an input stream of characters into a stream of tokens. While such programs are often best written using a *lexer generator* (*e.g.*, ML-ULex or Flex), for this assignment you will write a scanner from scratch.

2 Flang lexical conventions

Flang programs are written using the ASCII character set. The scanner is responsible for converting a sequence of ASCII characters (*i.e.*, a source file) into a sequence of *tokens*. There are five classes of tokens in **Flang**:

1. lower-case identifiers: `a`, `b`, `toString`, `y23`, *etc.*
2. upper-case identifiers: `X`, `Foo`, `SOME_VAL`, *etc.*
3. numbers: `0`, `42`, *etc.*
4. strings: `"hello␣world"`, `"some\ntext"`, *etc.*
5. delimiters and operators: `(`, `)`, `=`, `<=`, `+`, *etc.*

Tokens can be separated by *whitespace* and/or *comments*.

Type-variable, type-constructor, data-constructor, and value identifiers in **Flang** can be any string of letters, digits, underscores, and quote marks, beginning with a letter. Identifiers are case-sensitive (*e.g.*, `foo` is different from `Foo`). We use distinguish between identifiers that begin with an *upper-case* letter and those that begin with *lower-case* letters. We use upper-case identifiers for type and data constructors, and lower-case identifiers for type and value variables. The following lower-case identifiers are reserved as keywords:

```
case   con   data   else   end
 fun    if    let    of    then
type   with
```

Flang also has a collection of delimiters and operators, which are the following:

```
(   )   [   ]   {   }
=   ==  <=  <   :   ::
@   +   -   *   /   ,
;   ->  =>  _
```

Numbers in **Flang** are integers and their literals are written using decimal notation (without a sign).

String literals are delimited by matching double quotes and can contain the following C-like escape sequences:

`\a`	—	bell (ASCII code 7)
`\b`	—	backspace (ASCII code 8)
`\f`	—	form feed (ASCII code 12)
`\n`	—	newline (ASCII code 10)
`\r`	—	carriage return (ASCII code 13)
`\t`	—	horizontal tab (ASCII code 8)
`\v`	—	vertical tab (ASCII code 11)
`\\`	—	backslash
`\"`	—	quotation mark

A character in a string literal may also be specified by its numerical value using the escape sequence '*ddd*,' where *ddd* is a sequence of three decimal digits. Strings in **Flang** may contain any 8-bit value, including embedded zeros, which can be specified as '`\000`.'

Comments may start anywhere outside a string with "`(*`" and are terminated with a matching "`*)`". As in SML, comments may be nested.

Whitespace is any non-empty sequence of spaces (ASCII code 32), horizontal or vertical tabs, form feeds, newlines, or carriage returns. Any other non-printable character should be treated as an error.

3 Requirements

Your implementation should include (at least) the following two modules:

```
structure FLangLex : FLANG_LEXER
structure FLangTokens : FLANG_TOKENS
```

The signature of the `FLangLex` module is

```
signature FLANG_LEXER =
  sig
    val lexer : ((char, 'a) StringCvt.reader)
          -> (FLangTokens.token, 'a) StringCvt.reader
  end
```

The `StringCvt.reader` type is defined in the SML Basis Library as follows:

```
type ('item, 'strm) reader = 'strm -> ('item * 'strm) option
```

A reader is a function that takes a stream and returns a pair of the next item and the rest of the stream (it returns `NONE` when the end of the stream is reached). Thus, `lexer` is a function that takes a character reader and returns a token reader.

We will post a file `flang-tokens.sml` on Piazza that you may include in your project. It defines the `FLANG_TOKENS` signature, has the following form:

```
signature FLANG_TOKENS =
  sig
    datatype token
      = KW_case                 (* "case" *)
      | KW_con                  (* "con" *)
      | KW_data                 (* "data" *)
      | KW_else                 (* "else" *)
      | KW_end                  (* "end" *)
      | KW_fun                  (* "fun" *)
      | KW_if                   (* "if" *)
      | KW_let                  (* "let" *)
      | KW_of                   (* "of" *)
      | KW_then                 (* "then" *)
      | KW_type                 (* "type" *)
      | KW_with                 (* "with" *)
      | LP                      (* "(" *)
      | RP                      (* ")" *)
      | LB                      (* "[" *)
      | RB                      (* "]" *)
      | LCB                     (* "{" *)
      | RCB                     (* "}" *)
      | EQ                      (* "=" *)
      | EQEQ                    (* "==" *)
      | LTEQ                    (* "<=" *)
      | LT                      (* "<" *)
      | COLON                   (* ":" *)
      | DCOLON                  (* "::" *)
      | AT                      (* "@" *)
      | PLUS                    (* "+" *)
      | MINUS                   (* "-" *)
      | TIMES                   (* "*" *)
      | DIV                     (* "/" *)
      | COMMA                   (* "," *)
      | SEMI                    (* ";" *)
      | ARROW                   (* "->" *)
      | DARROW                  (* "=>" *)
      | WILD                    (* "_" *)
      | UID of Atom.atom        (* upper-case identifiers *)
      | LID of Atom.atom        (* lower-case identifiers *)
      | NUMBER of IntInf.int    (* integer literals *)
      | STRING of string        (* string literals; argument does not
                                 * include enclosing quotes *)
  end
```

The tokens correspond to the various keywords, delimiters and operators, and literals. The `LID` (*resp*. `UID`) token is for non-reserved lower-case (*resp*. upper-case) identifiers and carries a unique string representation of the identifier. The `NUMBER` and `STRING` tokens carry the value of the literal.

4 Submission

This project is due at 10pm on Friday, January 23rd. You should submit your code by committing it to your phoenixforge **svn** repository. Please put your code into a directory called "`proj1`."

5 Document history

January 14, 2015 Removed `EOF` token from description.

January 8, 2015 Original version.