1 Introduction

Your first assignment is to implement a lexer (or scanner) for MLR, 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-Lex or Flex), for this assignment you will write a scanner from scratch.

2 MLR lexical conventions

MLR has four classes of token: identifiers, delimiters and operators, numbers, and string literals. Tokens can be separated by whitespace and/or comments.

Identifiers in MLR can be any string of letters, digits, and underscores, not beginning with a digit. Identifiers are case-sensitive (e.g., foo is different from Foo). The following identifiers are reserved as keywords:

```mlr
andalso  bool  else  false  fun
hd       if     in     int     isnull
let      list   not    orelse  string
then     tl     true   type    with
```

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

```mlr
( ) { } [ ]
== <= < :: + -
* / % = . ,
; :
```

Numbers in RML are integers and are 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 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 MLR may contain any 8-bit value, including embedded zeros, which can be specified as `\000;`.

Comments 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:

```ml
structure MLRLexer : MLR_LEXER
structure MLRTokens : MLR_TOKENS
```

The signature of the MLRLexer module is

```ml
signature MLR_LEXER =

sig
  val lexer : ((char, 'a) StringCvt.reader) -> (MLRTokens.token, 'a) StringCvt.reader

end
```

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

```ml
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.

The signature of the MLRTokens module should have the following form:
signature MLR_TOKENS =

sig

datatype token =
  KW_andalso
  | KW_bool
  | KW_else...
  | KW_with
  | LP | RP
  | LCB | RCB (* '(' ')') *)
  | LSB | RSB (* '[' ']' *)
  | DEQ (* '==' *)
  | LTEQ | LT
  | DCOLON (* '::' *)
  | PLUS | MINUS | TIMES | DIV | MOD
  | EQ | DOT | COMMA | SEMI | COLON
  | NAME of Atom.atom
  | NUMBER of IntInf.int
  | STRING of string

end

The tokens correspond to the various keywords, delimiters and operators, and literals. The NAME token is for non-reserved identifiers and carries a unique string representation of the identifier. The NUMBER and STRING tokens carry the value of the literal.