Assignment #2: Lexical Analyzer

Due: January 22, 2010 11:59pm

Overview

This assignment requires you to implement a lexical analyzer (sometimes called a scanner or a tokenizer) for the simple language D$^\flat$ with which you will work for the remainder of the course. A scanner performs the first step in a compiler or interpreter; it reads characters from a file and produces tokens. In essence, the scanner is a finite state machine that produces each recognized "word" on acceptance (and then repeats in an attempt to recognize another "word").

Lexemes

The following is a list of the lexemes (words) that will be used when writing programs in the D$^\flat$ language.

- **Keywords**: `int bool fn write writeline if else while true false return var unit`
- **Assignment**: `:=`
- **Punctuation**: { } `( ) , ; ->`
- **Logical Operators**: `&` |
- **Relational Operators**: `=` $<$ $>$! $=$ `<=` `>=`
- **Arithmetic Binary Operators**: `+ - * /`
- **Unary Operators**: `!`
- **Numbers**: (`0` $\cup$ ... $\cup$ `9`)$^+$
- **Identifiers**: (`a` $\cup$ ... $\cup$ `z` $\cup$ `A` $\cup$ ... $\cup$ `Z`) (`a` $\cup$ ... $\cup$ `z` $\cup$ `A` $\cup$ ... $\cup$ `Z` $\cup$ `0` $\cup$ ... $\cup$ `9`)*

Or, as a (busy) regular expression:

`int` $\cup$ `bool` $\cup$ `fn` $\cup$ `write` $\cup$ `writeline` $\cup$ `if` $\cup$ `else` $\cup$ `while` $\cup$ `true` $\cup$ `false` $\cup$ `return` $\cup$ `var` $\cup$ `unit`
$\cup$ `=` $\cup$ { $\cup$ } $\cup$ `(` $\cup$ `)` $\cup$ `,` $\cup$ `;` $\cup$ `->` $\cup$ `&` $\cup$ | $\cup$ `=` $\cup$ $<$ $\cup$ $>$ $\cup$! $=$ $\cup$ `<=` $\cup$ `>=` $\cup$ `+` $\cup$ `-` $\cup$ `*` $\cup$ `/`
$\cup$ `!` $\cup$ (`0` $\cup$ ... $\cup$ `9`)$^+$ $\cup$ (`a` $\cup$ ... $\cup$ `z` $\cup$ `A` $\cup$ ... $\cup$ `Z`) (`a` $\cup$ ... $\cup$ `z` $\cup$ `A` $\cup$ ... $\cup$ `Z` $\cup$ `0` $\cup$ ... $\cup$ `9`)*

Notes

- Consider drawing a finite state machine for the above regular language if you think it will help in visualizing the language.
- As a simplifying step (and as is commonly done), you might treat all keywords as identifiers and distinguish between them once a full token is found.
- A token is formed by taking the longest sequence of valid characters. For example, the sequence of characters 'abc-123' will be recognized as the tokens 'abc', '-', and '123'.
- Whitespace serves only to delimit tokens.
- An end-of-file terminates tokenizing after the last token is recognized.
- If an illegal character is found, exit with `OS.Process.exit OS.Process.failure`.

Part 1: Recognizer

For this part of the assignment, you must write a function that takes an *instream* (typically bound to a file) and that reports, by printing, the first recognized token read from the input stream (stopping at the first character that is not a valid part of the matched token). Name your "main" function for this part `recognizeToken`. For the time being, the function can return the `unit` value (this will change for the next part). Once end-of-file has been reached, print "end-of-file" to the screen. (See the sample output for examples.)

Part 2: Tokenizer

Define a new datatype to represent the set of possible tokens. You must include constructors for each of the keywords and symbols as well as two parameterized constructors (one for numbers and one for identifiers). The constructor names for your datatype must be descriptive.

Starting with the above function(s) (make a copy of the code), write a function named `nextToken` that takes an *instream* and returns the next token on the stream (a value of the datatype that you defined above). If there are no remaining tokens, then return a special token value indicating end-of-file (the given testing function will assume that the constructor for this final value is named `TK_EOF`). (See the sample output for examples.) This function (or set of functions) must *not* print anything to the screen as it (they) will be used in a later assignment.

Functions

You might find some of the following functions to be of use.

TextIO.endOfStream	TextIO.lookahead	TextIO.input1
Char.isSpace	Char.isAlpha	Char.isDigit
isSome	valOf	TextIO.output
TextIO.print	OS.Process.exit	Int.fromString
TextIO.openIn		

You can open a module instead of prepending each function with the module name.

Logistics

- Strive for simplicity in your programming.
- Test data with "correct" output will be given. Your output can differ in minor ways from this "correct" output yet still be correct; it is up to you to verify your code's correctness.
- Grading will be divided as follows.

Part	Percentage
1	70
2	30

- Get started **now** to avoid the last minute rush.