CS305 – Programming Languages
Spring 2019-2020

Homework 1

Due date: February 21, 2020 @ 23:55

Implementing a Lexical Analyzer (Scanner) for BMO

1 Introduction

In this homework you will implement a scanner for ***BMO*** (which stands for "basic math operations") language using flex. Your scanner will be used to produce the tokens in a ***BMO*** program. In a ***BMO*** program, there may be keywords (e.g. vector, if, ...), numbers (both integer and real, both negative and positive), identifiers (programmer defined names for variables, etc.), operators and punctuation symbols (like + ; etc.). Your scanner will catch these language constructs (introduced in Sections 2, 3, 4, 5) and print out the token names together with their positions (explained in Section 6) in the input file. Please see Section 7 for an example of the required output. The following sections provide extensive information on the homework. Please read the entire document carefully before starting your work on the homework.

2 Keywords

Below is the list of keywords that will be implemented. The list corresponds to the complete list of BMO standard keywords.

int	real	if	endif	vector	matrix	transpose

We will assume that the token name for a keyword is formed by appending upper case keyword to t. For example, for the keywords vector and if, we

will use the token names ***tVECTOR*** and ***tIF***, respectively.

3 Operators & Punctuation Symbols

Below is the list of operators and punctuation symbols to be implemented, together with the corresponding token names.

Lexeme	**Token**	**Lexeme**	**Token**
,	tCOMMA	.*	tDOTPROD
(	tLPAR	)	tRPAR
=	tASSIGNM	-	tMINUS
+	tPLUS	/	tDIV
*	tSTAR	;	tSEMI
==	tEQUL	!=	tINEQ
<	tLT	<=	tLE
>	tGT	>=	tGE
[	tLBRAC	]	tRBRAC
&&	tAND	||	tOR

4 Identifiers & Numbers

You need to implement identifiers and numbers. The rule for an identifier is the following:

- An identifier consists of any combination of letters, digits and underscore character. However, it cannot start with a digit.
- The token name for an identifier is tIDENT.
- You need to implement positive and negative integers and reals.

- A positive integer number consists of a sequence of digits optionally followed by an exponent (denoted by E), which is defined by a sequence of digits. Note that exponent cannot be negative for integers. The token to be used for positive integers is tPOSINT. An integer number can start with zeros, hence 0012 should infer a tPOSINT token.

- A negative integer is the same as a positive integer, however it starts with a minus sign. The token to be used for negative integers is tNEGINT. Note that, -123 must infer tNEGINT, however - 123 (with a space between - and 123) must infer one tMINUS (see Section 3) and one tPOSINT tokens.
  Examples of positive & negative integers.

  12

  -23

  -012002

  0050E230

  -105E134

- Real numbers are defined in a similar way with integer numbers, the only difference is that sequence of digits prior to optional exponent has a dot in it (see below for examples). However, there always exists at least one digit before the dot. Note that exponent can be negative for real numbers as seen in the last example given below. The exponent itself will always be an integer. Tokens for real number are tPOSREAL and tNEGREAL.

  0.0

  -00120.200200

  0011.2311E-000450

- Apart from scalars, you need to implement matrix and vectors as well. A vector or a matrix can hold integers or reals. A vector is a one dimensional data structure that may have many items of the same type. A vector value is given by providing the comma separated list of the items surrounded by square brackets.

A matrix has two dimensions storing items of the same type where rows are separated by semicolons. Each row is given by providing the comma separated list of the items.

Examples of vector & matrix data structures:

```
int vector myVector = [1, 3, 5, 7, 99, 7, 44, 3];
int vector myVector_2;
int matrix myMatrix = [1, 3; 5, 7; 99, 7; 44, 3];
int matrix myMatrix_2;
```

- Moreover, for all identifiers and numbers you need to store the actual lexemes associated with the tokens. You are required to output the lexemes for the identifiers and the positions of these lexemes. For numbers you will output both the lexemes and the corresponding values along with their positions in your output. Note the lexeme and the value of a number may not be the same. For example a number with the lexeme 0012 has the value 12. See Section 7 for details on the output to be produced.

5 Comments

A comment starts with // and extends up to the end of the line. A comment may appear on any line of a program. Anything that is commented out will be eaten up by your scanner silently. That is, no information will be produced for the comments or for the contents of the comments.

Examples of comments:

```
// this is an declaratjon and initialization of a vector
int vector myVector = [1, 3, 5, 7, 99, 7, 44, 3];
int vector myVector_2; // a declaration without an initialization
// some matrix examples
int matrix myMatrix = [1, 3; 5, 7; 99, 7; 44, 3];
int matrix myMatrix_2;
```

6 Positions

You must keep track of the position information for the tokens. For each token that will be reported, the line number at which the lexeme of the token appears is considered to be the position of the token. Please see Section 7 to see how the position information is reported together with the token names.

7 Input, Output, and Example Execution

Assume that your executable scanner (which is generated by passing your flex program through flex and by passing the generated lex.yy.c through the C compiler gcc) is named as BMOscanner. Then we will test your scanner on a number of input files using the following command line:

`BMOscanner < test17.bmo`

As a response, your scanner should print out a separate line for each token it catches in the input file (test17.bmo given in Figure 1). The output format for a token is given below for each token separately:

Token	**Output**
identifier, number	⟨*row*⟩⟨*space*⟩⟨*token_name*⟩⟨*space*⟩(⟨*lexeme*⟩)
for all other tokens	⟨*row*⟩⟨*space*⟩⟨*token_name*⟩

```
if no > max
        // a bigger number is seen
        max = no;
endif;
```

Figure 1: An example BMO program: test17.bmo

Here, ⟨*row*⟩ gives the location (line number) of the first character of the lexeme of the token and ⟨*token_name*⟩ is the token name for the current item. ⟨*lexeme*⟩ will display the lexeme of the tokens of type identifier (tIDENT) and number (i.e. tNEGINT, tPOSINT, tNEGREAL, tPOSREAL). And ⟨*space*⟩ corresponds to a single space. For example let us assume that test17.bmo has the following content:

Then the output of your scanner must be:

```
1 tIF
1 tIDENT (no)
1 tGT
1 tIDENT (max)
3 tIDENT (max)
3 tASSIGNM
3 tIDENT (no)
3 tSEMI
4 tENDIF
4 tSEMI
```

Note that, the content of the test files need not be a complete or correct BMO programs. If the content of a test file is the following:

x34_5 -00.03750

if

matrix [0015E0012 , -1.20E-002 , 12.500]

Then your scanner should not complain about anything and output the following information:

```
1 tIDENT (x34_5)
1 tNEGREAL (-00.03750) (-0.0375)
2 tIF
3 tMATRIX
3 tLBRAC
3 tPOSINT (0015E0012) (15E12)
3 tCOMMA
3 tNEGREAL (-1.20E-002) (-1.2E-2)
3 tCOMMA
3 tPOSREAL (12.500) (12.5)
3 tRBRAC
```

8 How to Submit

Submit only your flex file (**without zipping it**) on SUCourse. The name of your flex file must be:

username-hw1.flx

where username is your SuCourse username.

9 Notes

- **Important**: Name your file as you are told and **don't zip it.**

  [-10 points otherwise]

- Make sure you print the token names exactly as it is supposed to be.
- No homework will be accepted if it is not submitted using SUCourse.
- You may get help from our TA or from your friends. However, **you must write your flex file by yourself**.
- Start working on the homework immediately.
- No late submission will be accepted.
- Since the grading will be done automatically on the flow.sabanciuniv.edu machine, we strongly encourage you to at least test your code on flow before submitting it.