COMP 3031 Assignment 2

Flex and Bison Programming

Fall 2017

Due: 5:00 PM on Nov 10, Friday

Instructions

- In this assignment, you will add code at the specified places in the two files, *matfunc.lex* and *matfunc.y*, in the folder *assign2*.
- To test your program, you can run the file *run.sh* in the folder *assign2*, the command is:
  sh run.sh
  If your program is correct, the content of file *results.txt* will be the same as that of the file *Expected_results.txt*.
- When you are ready to submit your work, zip the entire folder *assign2* and submit the zip file through Canvas.
- A significant number of points will be deducted if your code has compilation errors.

I. Problem Description

Your assignment is to use Flex and Bison to implement an integer matrix function calculator. Specifically, you will add Flex and Bison rules at the specified places in the two given code skeleton files, *matfunc.lex* and *matfunc.y*, so that the generated program will take integer matrix expressions as input, and evaluate the result following operator precedence and associativity. All matrices are two-dimensional.

Some examples are as follows:

Input: [[1]]+[[1]]

Output: [[2]]

It computes the element-wise sum of matrix [[1]] and matrix [[1]], which is [[1+1]] = [[2]].

Input: [[1,2], [4,5], [6,7]]   +[[1,2], [8,9], [4, 3]]

Output: [[2,4],[12,14],[10,10]]

It computes the element-wise sum of two matrices. The shape of the matrix [[1,2], [4,5], [6,7]] is 3x2, i.e. 3 rows and 2 columns. [1,2] is the first row of this matrix.

Input: REV [[1,2,3],[4,5,6]]

Output: [[6,5,4],[3,2,1]]

It outputs the elements of matrix[[1,2,3],[4,5,6]]in the reversed order.

Input: NEG([[1,2,3],[1,2,3]] + [[1,2,3],[1,2,3]]   )

Output: [[-2,-4,-6],[-2,-4,-6]]

It first computes the sum of matrix [[1,2,3],[1,2,3]]and [[1,2,3],[1,2,3]]. Then it outputs the negation of every element in the sum matrix.

II. Input Tokens

(a) Integer matrix

An integer matrix is surrounded by a pair of brackets "[" and "]". With the pair of brackets, a row of matrix is also surrounded by a pair of brackets "[" and "]". The integers in each row are separated by ","s. The following are examples:

(1) [[1,2,3],[4,5,6],[7,8,9]] represents the 3x3 matrix:

```
[   [   1   2   3]
    [   4   5   6]
    [   7   8   9]   ]
```

(2) [[1]] is a 1x1 matrix with the only element being 1.

(3) [[1,2,3]] is a 1x3 matrix having only one row which is [1,2,3].

(4) Integers can be positive, zero, or negative.

Empty matrix [[]] is not allowed.

(b) Operator

There are 7 operators, "+", "-", "*", "/", "REV", "NEG", and the parentheses "( )". Each of the operators "+", "-", "*", "/" performs the element-wise computation between two matrices of the same size (number of rows and number of columns).

III. Precedence and Associativity

The operators are listed in the order of decreasing precedence.

( )		
REV	NEG	(right associative)
*	/	(left associative)
+	-	(left associative)

For example:
Input: [[2,4],[6,8]] + [[2,4],[6,8]] / [[2,2],[2,2]]
Output: [[3,6],[9,12]]

Input: ([[2,4],[6,8]] + [[2,4],[6,8]]) / [[2,2],[2,2]]
Output: [[2,4],[6,8]]

IV. BNF of the Input

```
<S1> ::= <S1>+<S2> | <S1>-<S2> | <S2>
<S2> ::= <S2> * <S3> | <S2>/<S3> | <S3>
<S3> ::= REV<S3> | NEG<S3> | <S4>
<S4> ::= <MATRIX> | (<S1>)
<MATRIX> ::= [<COLS>]
<COLS> ::= <ROW> | <ROW>,<COLS>
<ROW> ::= [VEC]
<VEC> ::= <NUM>,<VEC> | <NUM>
<NUM>::= <M><D> | <M><ND><N>
<N>::= <D><N> | <D>
<D>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<ND>::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<M>::= -|<empty>
```

V. Remarks

1. You only need to add code in the three specified places:

(1) in *matfunc.lex*:

```
/************* Start: add your definitions here */

/* End: add your definitions here */
```

(2) in *matfunc.lex*:

```
/*		Start: add your rules here		*/

/*		End: add your rules here		*/
```

(3) in *matfunc.y*:

```
// start of your grammar rules and actions


// end of your grammar rules and actions
```

(4) The tokens defined in the code skeleton is sufficient for the assignment:

```
%token LB RB MAT ADD MINUS DIV NEG REV DOT
```

There may be other solutions that use a subset of them, or use other tokens. In case you want to define your own tokens, you can modify this line.

Besides the above-mentioned parts, DO NOT change any code in any area.

2. When testing your code, you can assume that the input is correct.

3. In your code, when you copy the value from yytext to yylval, you can use the following:

```
yylval = (char*)malloc(sizeof(char)*MAXL);
strcpy(yylval, yytext);
```

4. Functions you need to use:

```
char* Add(char *str1, char *str2);
char* Minus(char *str1, char *str2);
char* Dot(char *str1, char *str2);
char* Div(char *str1, char *str2);
char* Neg(char *str);
char* Rev(char *str);
char *FFormat(const char* str);
```