COMP 3031 Assignment 2
Flex and Bison Programming

Fall 2020

Due 5pm, 29th Oct. 2020 Thursday

1. Problem Description

In this assignment, you will implement a matrix calculator by filling in the blanks in the provided flex and bison code skeletons.

A matrix is of the form:

a11, a12, a13, …, a1m

a21, a22, a23, …, a2m

…

an1, an2, an3, …, anm

We use the following format in our matrix calculator to represent this matrix:

[a11, a12, a13, …, a1m; a21, a22, a23, …, a2m; …; an1, an2, an3, …, anm]

In this assignment, we assume all the elements of input matrix are **non-negative integers**.

The following is the BNF grammar rules for the matrix:

```
<matrix> ::= <left_square_bracket> <rows> <right_square_bracket>

<rows> ::= <rows> <semicolon> <row> | <row>

<row> ::= <row> <comma> <element> | <element>

<element> ::= <integer>

<integer> ::= <digit>|<digit><integer>

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<left_square_bracket> ::= [

<right_square_bracket> ::= ]

<semicolon> ::= ;

<comma> ::= ,
```

The matrix calculator evaluates expressions with operators on matrices, including the addition "+", the subtraction "-", and the multiplication "*". All three operators are left associative. The orders of precedence of these operators together with the parentheses "()". are listed in the following table:

Operators	Operations	Associativity	Precedence
+, -	Addition, Subtraction	Left	Low
*	Multiplication	Left	Medium
()	Parentheses	NA	High

The following BNF grammar rules define expressions with matrix operations:

```
<expression> ::= <expression> <addition> <sub_expression>

                 | <expression> <subtraction> <sub_expression>

                 | <sub_expression>

<sub_expression> ::= <sub_expression> <multiplication> <unit>

                 | <unit>

<unit> ::= <left_circle_bracket> <expression> <right_circle_bracket>

                 | <matrix>

<addition> ::= +

<subtraction> ::= -

<multiplication> ::= *

<left_circle_bracket> ::= (

<right_circle_bracket> :: )
```

2. Your work

We provide a zip package containing the following files:

assignment2.pdf	Assignment 2 description
helpers.h, helpers.c	Helper functions
Makefile	Makefile that supports "make" and "make clean"
matcal.lex	Flex code skeleton for you to fill in
matcal.y	Bison code skeleton for you to fill in
matcal-example	An example executable file

You can decompose the zip file using the following command

```
    unzip assignment2-2020fall.zip
```

You need to add **missing flex definitions and rules** in *matcal.lex* and **missing tokens and grammar rules** in *matcal.y*.

We have marked the blanks that you can fill in with comment lines:

```
/********** Start: ... **********/
```

```
/********** End: ... *********/
```

Note that do **not** modify the provided files *Makefile, helpers.h* and *helpers.c.*

After finishing the code, you can compile your code on a CS Lab 2 machine with the command

```
make
```

After successful compilation, you can run the binary, and type in the commands to start the matrix calculator:

```
./matcal
```

Your result matrix calculator will work as follows:

(1) If the input expression is a single matrix, your calculator will print the matrix. For example,

```
$ ./matcal
[9,8,7]
9     8     7
$ ./matcal
[1,2,3;4,5,6]
1     2     3
4     5     6
```

(2) If the input expression contains operators, your calculator will compute the result of the expression and print the output matrix. For example,

```
$ ./matcal
[2,3,4;2,3,4]+[4,5,6;4,5,6]
6     8     10
6     8     10
$ ./matcal
([1;2;3]+[6;5;4])*[7,8,9]
49    56    63
49    56    63
49    56    63
```

matcal-example is the example executable file and you can run `./matcal-example` for further information. Your program does **not** need to handle input errors.

3. Helper functions

We list the helper functions in *helpers.h* and *helpers.c* in the following table:

Helper function	**Description**
void print_matrix(void *payload);	Print a matrix payload
void *append_row(void *payload1, void *payload2);	Append a 1*m matrix payload2 to the end of an n*m matrix payload1, return the result matrix
void *append_element(void * payload1, void *payload2);	Append a 1*1 matrix payload2 to the end of a 1*m matrix payload1, return the result matrix
void *element2matrix(int e);	Convert an integer to a 1*1 matrix, return the result matrix
void *matrix_add(void *payload1, void *payload2);	Matrix addition, return the result matrix
void *matrix_sub(void *payload1, void *payload2);	Matrix subtraction, return the result matrix
void *matrix_mul(void *payload1, void *payload2);	Matrix multiplication, return the result matrix

We use pointers of void* type to point to a matrix. You do **not** need to do any type conversion in your assignment.

4. Submission

- Zip your two source files, *matcal.lex* and *matcal.y*, into a single package using exactly the following command (case-sensitive):

  ```
  zip matcal.zip matcal.lex matcal.y
  ```

- Submit your zipped file *matcal.zip* to Canvas.
- **No late submission will be accepted.**
- Your submission will be compiled and run on a CS Lab 2 machine. If it cannot be compiled or run, you may get 0 marks for this assignment.
- We will use tools to detect code similarity. On confirmed cases of high code similarity, we will follow university guidelines on academic integrity to take necessary actions.