CS39003 Compilers Laboratory
Autumn 2026
Assignment 5
Date of posting: 25-Aug-2026

Introduction to yacc

In this assignment, you continue to use the language of matrix assignments. A new operator # is now introduced. When applied *after* a matrix variable or a parenthesized matrix expression, it transposes the matrix. Since an LALR(1) parser will be generated by yacc, it is not necessary to avoid left recursion. In view of this, the grammar for matrix assignments is rewritten as follows. Henceforth the terminal symbols will be named in lower case.

PROG	⟶	LINE **\n** | PROG LINE **\n**	The entire input
LINE	⟶	**id** **=** RVAL	A single matrix assignment
RVAL	⟶	**[** ROWLIST **]** | EXPR	R-value in an assignment
ROWLIST	⟶	ROW | ROWLIST **;** ROW	List of rows of a literal matrix
ROW	⟶	**num** | ROW **,** **num**	A single row in a literal matrix
EXPR	⟶	TERM | EXPR **+** TERM | EXPR **–** TERM	Matrix expression
TERM	⟶	FACTOR | TERM ***** FACTOR	A term in a matrix expression
FACTOR	⟶	SCALAR MATFAC TRANS	A factor in a term
SCALAR	⟶	ε | **num**	Optional scalar multiplier
TRANS	⟶	ε | **#**	Optional transpose operator
MATFAC	⟶	**id** | **(** EXPR **)**	The core of a factor

Write the following files.

- **A lex file**

  Your lex file will keep on *returning* tokens to yacc. In this grammar, the tokens are **id** (matrix names), **num** (integers with an optional **+** or **–** sign), operators **=**, **+**, *****, and **#**, punctuation symbols (left and right parentheses, left and right brackets, comma, and semicolon), the new-line character **\n**, and **err** (an unmatched character).

  In this assignment, allow expressions of the form `M1-2M2` or `M1-3(M2+2M3*M4)` without any space between the `+` or the `-` operator and a following scalar multiplier. But lex will club together the sign with the immediately following unsigned scalar multiplier. (In a literal matrix, signed numbers do not create any confusion.) Use lex and yacc carefully (change the tokens and the grammar appropriately) to solve this problem.

  The lex portion of the solution can be like this. Instead of returning the token **num**, lex will return separate tokens **unum** (unsigned numbers) and **snum** (signed numbers). Yacc will handle them appropriately.

- **A yacc file**

  Use the above grammar in your yacc file. You would need small changes to handle signed integers (as explained in connection with lex). For example, `M1-2M2` will let lex generate the tokens EXPR **snum** TERM. There is however no production in the grammar of this form. So yacc should interpret the expression as EXPR **+** TERM. The **snum** will be shifted to TERM as the scalar multiplier.

  Your yacc actions must not implement any matrix operation or symbol-table utility. The yacc actions will only call the appropriate functions defined elsewhere (see below).

  Write a `main()` function in your yacc file. So `y.tab.c` will be the main compilable file that will include (or link during compilation) the other source files (including `lex.yy.c`).

- **A source file for matrix utilities**

  Define a suitable data type for matrices. References to matrices will reside in the yacc stack. The stack element can be of basic C types (like `char`, `int`, or `double`) or pointers to basic C types or to `struct` types. A C++-specific type (like a class) for storing a matrix cannot be used. Write all matrix-related operations in this file.

- **A source file for symbol-table management**

  The organization of the symbol table (to be maintained globally) is left to you. Since symbol-table entries will not reside in the yacc stack, you can use C++-specific features to implement your symbol table. Write the relevant utilities (search, add/update/get entries, print entries) in this file.

- **Any other source/header files (optional)**

  If needed, you may write the type definitions and function-prototype declarations (and possibly global variables too) in a separate source/header file.

- **A makefile**

  Write a makefile with compile, run, and clean targets. The run target may assume that the interpreter is fed from the text file `input.txt`. Set the input for lex (`yyin`) accordingly. Use the clean action before submitting.

Combine all the above files into a single tar/tgz/zip archive, and submit that archive only.

Sample Output

Input

```
M1 = [ 1,2,3,4; 5,6,7,8; 9,10,11,12 ]

M2 = [ -1,2,-3,4; 5,-6,7,-8; -9,10,-11,12 ]

M3 = [ 1,2,3; 4,5,6; 7,8,9; 10,11,12 ]

M4 = M1 * M3
M5 = 2M2 * 3M3

M6 = M3 * (M1 + 2M2)
M7 = -2M3 * (M1 - 2M2) + M6

M8 = -2(M3*-6(M1-2M2)+2(2M6-5M7)*2M6)

M9 = 2 ( ( M1 + M2 - 2 M1 ) * M6 # * M3 + 100 M5 + 1000 M4 )

M10 = 2(3M1 * 4M2#)
M10 = 2(3M1 * 4M2#)#

M11 = ( (M1-2M2) * M3 * 2(M1-M2)-M1 ) * (M5-M4)#
M11 = (M5-M4)# * ( (M1-2M2) * M3 * 2(M1-M2) - M1 )

M12 = (((M1*-2M1#) + (M2*3M2#)) * (M2-M3#))#
```

Output

```
M1  =              1           2           3           4
                   5           6           7           8
                   9          10          11          12

M2  =             -1           2          -3           4
                   5          -6           7          -8
                  -9          10         -11          12

M3  =              1           2           3
                   4           5           6
                   7           8           9
                  10          11          12

M4  =             70          80          90
                 158         184         210
                 246         288         330

M5  =            156         168         180
                -300        -312        -324
                 444         456         468

M6  =              2          84           6         104
                  17         174          27         224
                  32         264          48         344
                  47         354          69         464

M7  =           -146          76        -182          88
                -281         130        -371         160
                -416         184        -560         232
                -551         238        -749         304

M8  =         -130824    -1488336     -214296    -1906848
              -336516    -3581304     -540204    -4602624
              -542208    -5674272     -866112    -7298400
              -747900    -7767240    -1192020    -9994176

M9  =          154464      174672      194880
               -149568     -157344     -165120
                503136      579408      655680

M10 =             240        -432         624
                  432        -624         816
                  624        -816        1008

M10 =             240         432         624
                 -432        -624        -816
                  624         816        1008
*** Incompatible matrices for multiplication

M11 =   EMPTY

M11 =        -4298458    -2469292    -5944702    -3292240
             -4969616    -2828192    -6855344    -3770816
             -5640774    -3187092    -7765986    -4249392

M12 =           -2430       17902       -9742
                 4230       -6774       16854
                -2150       31126       -8630
                 6470       -1206       25750
```