CARLETON UNIVERSITY Department of Systems and Computer Engineering
SYSC 3101 Programming Languages
Assignment 2 Parsing and Lexical Analysis Midnight Sunday, March 13^{th}.

Syntax Analysis – LR Parser

10 marks

Consider the classic grammar for arithmetic expressions:

1. $E \rightarrow E + T$ 2. $E \rightarrow T$ 3. $T \rightarrow T * F$
4. $T \rightarrow F$ 5. $F \rightarrow (E)$ 6. $F \rightarrow id$

and the LR parsing table below where S*i* means shift and stack state i, R*j* means reduce by production numbered j, and blank means error. Using the grammar and the LR parsing table, show a complete parse, including the parse stack contents, input string, and action for the string `(id + id) * id + id`

	Action						Goto		
State	ID	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R3	R5		R5	R5			

A simple expression analyzer

20 marks

Extend the grammar shown above to implement exponentiation, y^x, PLUS the functions `log(x)` and `exp(x)` for the natural log of x and e^x. Use the circumflex, '`^`', for your exponentiation operator. Also: exponentiation is *right*-associative and has a higher precedence than mutliplication and division.

Write a program using Flex, Bison, and C to implement your grammar. You may use Lex in place of Flex and Yacc in place of Bison. You can use any C compiler of your choosing (Borland C and Visual C have both been used in the lab.) Using Linux instead of Windows may also be less painful. The input of the program consists of the expression to be evaluated and the output is the result. Use only integer or floating point constants for `CONST`. The lexical analyzer should convert a floating point number into a double (or float) so it should recognize exponents. You can either parse unary + and −, or handle it during lexical analysis. Any other input should generate an error.

Hints:

- Use `sscanf()`, `atof()`, or `strtod()` in your lexical analyzer to convert a string to a floating point number.
- Have your lexical analyzer discard WHITE SPACE such as spaces, carriage returns and new lines (otherwise, your parser will have to process it!)
- Use '`#define YYDEBUG 1`' and set the variable '`yydebug = 1;`' to enable debug output.

Please see the course web site for Flex/Lex/Bison/Yacc information.