CISP 430
Data Structures
Assignment 3

Expression Evaluation
Statement of Problem

The ability to evaluate numeric expressions containing constants, variables, and function references is basic to all programming languages. Evaluation of numeric expressions involves techniques which convert the expression into some "internal" form.

The technique described in this assignment involves the use of a stack and queue mechanisms, the postfix expression notation form, and a postfix expression evaluator. Write a program capable of evaluating expressions of the following general forms:

```
A = A + B / C * D
B = 5 / 2.00 + A
C = SIN(A) + (C-D) * 3
D = A - B * C / D
```

Expressions may be entered via keyboard or a disk file. Variables A, B, C, and D are single character variables having fixed values or values derived via prior assignment. The functions SIN,COS,ABS, and SQRT are also valid terms within an expression. The actual value of any variable could be integer or real. The value of a variable is considered only when reducing the right-hand side of the expression to a single value (see function GetValue on next page).

The standard rules of operator precedence apply; division and multiplication occur before addition and subtraction, which occur before any assignment operation. You will not actually implement the assignment (=) operation; simply output the expression "var = thevalue". An example follows assuming that A=5, B=10, C=-1, and D=2.

```
D = D*C-A/B reduces to
D = -2.5
```

If fixed values are used for variables 'A'-'D', the Pascal function GetValue returns the values of variables 'A' - 'D' when invoked. Use this function to reduce the right-hand side (rhs) of an expression to a single value during expression evaluation.

```
typedef int valuetype;

valuetype GetValue(char variable){
  switch variable {
    case 'A': return avalue;break;
    case 'B': return bvalue;break;
    case 'C':  ....
    case 'D':  ....
  }
} // GetValue
```

NextState Pushdown Transducer

		Input Symbol						
		<identifier>	=	+,-	*,/	(	)	End
Top of Stack S2	Null	S_1	S_2	Err	Err	Err	Err	Err
	=	S_1	Err	S_2	S_2	S_2	Err	U_2
	+,-	S_1	Err	U_1	S_2	S_2	U_c	U_2
	*,/	S_1	Err	U_1	U_1	S_2	U_c	U_2
	(	S_1	Err	S_2	S_2	S_2	U_c	U_2

Source: Applying Data Structures, T.G. Lewis (Univ. SW Louisiana) and M.Z. Smith(IBM), Houghton-Mifflin, 1976

S_1 - Stack input onto S1.
S_2 - Stack input onto S2.
Err - error occurred, input not valid.
U_1 - unstack S2 --> S1, do another comparison.
U_c - Unstack S2 --> S1 until "(" is found; discard "(".
U_2 - Unstack S2 --> S1 until S2 is empty.