test\validarithmeticexpression.y

```
LEX PART:
%{
    #include<stdio.h>
    #include "y.tab.h"
%}

%%
[a-zA-Z]+ return VARIABLE;
[0-9]+ return NUMBER;
[\t] ;
[\n] return 0;
. return yytext[0];
%%

int yywrap()
{
    return 1;
}







YACC PART:
%{
    #include<stdio.h>
%}
%token NUMBER
%token VARIABLE

%left '+' '-'
%left '*' '/' '%'
%left '(' ')'

%%
S: VARIABLE'='E {
       printf("\nEntered arithmetic expression is Valid\n\n");
       return 0;
     }
E:E'+'E
 |E'-'E
 |E'*'E
 |E'/'E
 |E'%'E
 |'('E')'
 | NUMBER
 | VARIABLE
;
%%
```

```

void main()
{
   printf("\nEnter Any Arithmetic Expression:\n");
   yyparse();
}

void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");

}






ALGORITHM:
1. Start

2. Lex Part (Tokenization):
    2.1 Define tokens based on lexical rules:
        2.1.1 [a-zA-Z]+: Matches variables; return the VARIABLE token
        2.1.2 [0-9]+: Matches numeric constants; return the NUMBER token
        2.1.3 [\t]: Matches tabs and ignores them
        2.1.4 [\n]: Matches newline and returns 0 to indicate the end of input
        2.1.5 .   : Matches any other character and returns it
    2.2 Define yywrap() function:
        2.2.1 Called when the input is exhausted.
        2.2.2 Return 1 to indicate EOF.
    2.3 Output tokens to YACC for parsing.

3. YACC Part (Parsing):
    3.1 Receive tokens from Lex.
    3.2 Set operator precedence and associativity
    3.3 Parse token sequence based on defined grammar rules:
        3.3.1 S: Starting rule:
                - VARIABLE '=' E: Print "Entered arithmetic expression is valid" and return 0 when a valid expression is recognized
        3.3.2 E: Expression rule with the following sub-rules:
                - E '+' E: For addition
                - E '-' E: For subtraction
                - E '*' E: For multiplication
                - E '/' E: For division
                - E '%' E: For modulus
                - '(' E ')': For expressions within parentheses
                - NUMBER: For numeric values.
                - VARIABLE: For variable values
    3.4 Implement yyerror() function:
        3.4.1 Print "Entered arithmetic expression is invalid" when parsing fails

4. main() Function:
    4.1 Prompt the user to enter an arithmetic expression
    4.2 Invoke yyparse() to begin parsing the input

5. Stop
```