Exp 8b : YACC – VALID EXPRESSION (C)

LEX

1. Start
2. [0-9]+ : matches one or more digits and convert value of yylval to the integer representation of the matched text and returns the token type NUM
3. [a-zA-Z][a-zA-Z0-9]* : matches an identifier and returns the token type ID
4. [\t\n] : skips whitespaces
5. . : matches any other character and returns it as a single character token
6. yywrap() indicate the end of input
7. Stop

YACC

1. Start
2. Token Declarations (%token) : NUM and ID
3. Grammar Rules Section

   S -> ID = E ;

   E -> E+E | E-E | E*E |E/E | E%E | (E) | ID | NUM | (ID) | (NUM)
4. yyerror() to handle
5. main() prompts the user to enter an arithmetic expression and calls yyparse() to parse it
6. Stop

Valid C Expression (Lex)

```
%{
   #include "y.tab.h"
%}

%%
[a-zA-Z][a-zA-Z0-9]* { return ID; }
[0-9]+ { return NUM; }
[ \t\n] {;}
. { return yytext[0]; }
%%

int yywrap(){
   return 1;
}
```

Valid C Expression (YACC)

```
%{
   #include<stdio.h>
   int flag = 0;
%}

%token NUM ID

%%
start : ID '=' E ';' ;
nid : ID | NUM ;
E : E'+'E
| E'-'E
| E'*'E
| E'/'E
| nid
| '(' nid ')'
;
%%

int yyerror(){
   flag = 1;
   printf("Invalid expression\n");
   return 0;
}

int main(){
   printf("Enter the expression : ");
   yyparse();
   if(flag == 0){
      printf("Valid expression\n");
   }
   return 0;
}
```

Note : if output is not printed then press ctrl+D