Ex No: 5 **Date:**

RECOGNIZE AN ARITHMETIC EXPRESSION USING LEX AND YACC

AIM:

To check whether the arithmetic expression using lex and yacc tool.

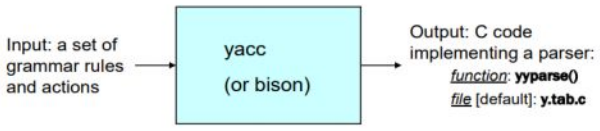
ALGORITHM:

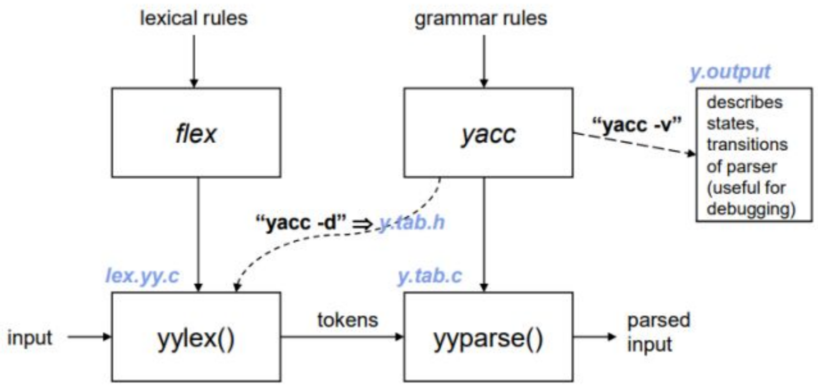
- Using the flex tool, create lex and yacc files.
- In the C include section define the header files required.
- In the rules section define the REGEX expressions along with proper definitions.
- In the user defined section define yywrap() function.
- Declare the yacc file inside it in the C definitions section declare the header files required along with an integer variable valid with value assigned as 1.
- In the Yacc declarations declare the format token num id op.
- In the grammar rules section if the starting string is followed by assigning operator or identifier or number or operator followed by a number or open parenthesis followed by an identifier. The x could be an operator followed by an identifier or operator or no operator then declare that as valid expressions by making the valid stay in 1 itself.
- In the user definition section if the valid is 0 print as Invalid expression in yyerror() and define the main function.

LEX AND YACC WORKING :

Parser generator:

- Takes a specification for a context-free grammar.
- Produces code for a parser.


Input: a set of grammar rules and actions
yacc
(or bison)
Output: C code implementing a parser:
function: yyparse()
file [default]: y.tab.c



PROGRAM:

validexp.l:

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

%%
[a-zA-Z]+ return VARIABLE;
[0-9]+ return NUMBER;
[\t] ;
[\n] return 0;
. returnyytext[0];
%%
intyywrap()
{
return 1;
}
```

validexp.y:

```
%{
   #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
;
```

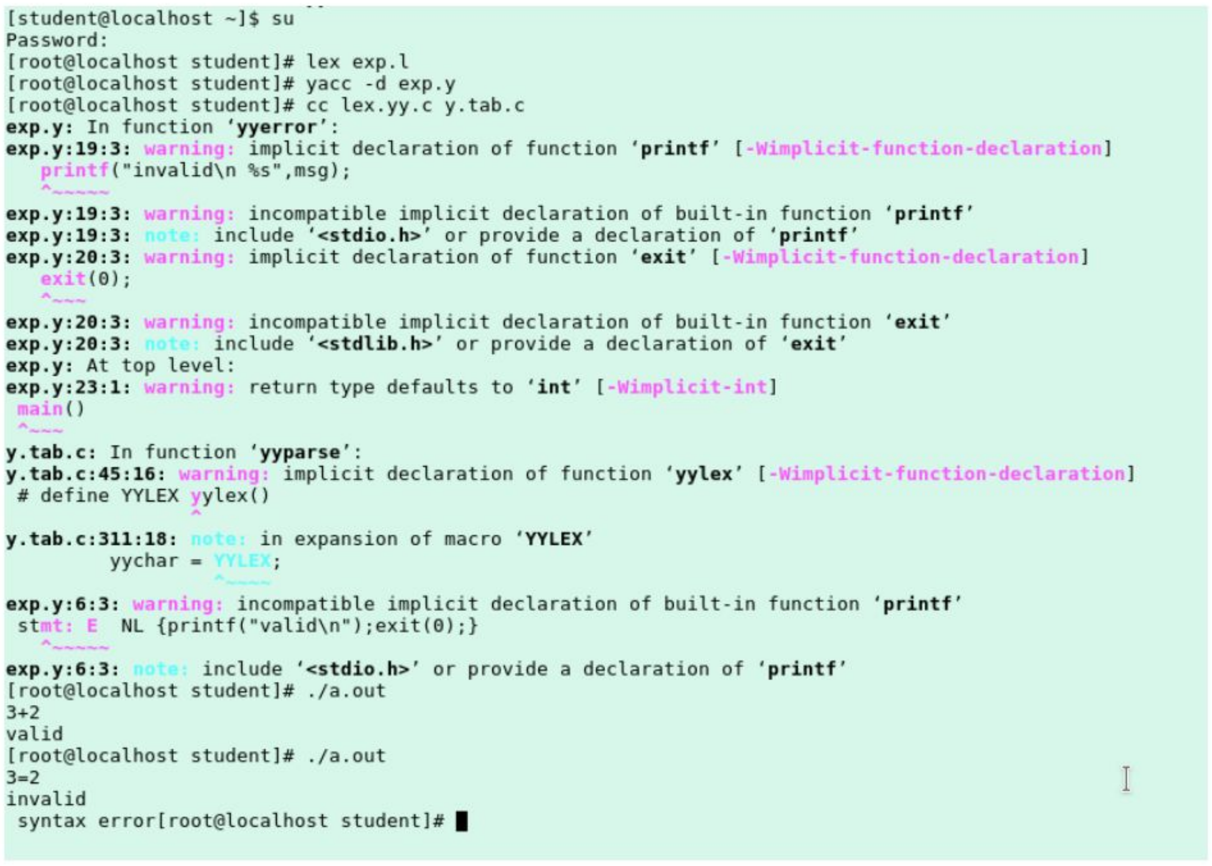
```
%%

void main()
{
  printf("\nEnter Any Arithmetic Expression which can have operations Addition,
Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");
  yyparse();
}

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

}
```

OUTPUT:



```
[student@localhost ~]$ su
Password:
[root@localhost student]# lex exp.l
[root@localhost student]# yacc -d exp.y
[root@localhost student]# cc lex.yy.c y.tab.c
exp.y: In function ‘yyerror’:
exp.y:19:3: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
   printf("invalid\n %s",msg);
   ^~~~~~
exp.y:19:3: warning: incompatible implicit declaration of built-in function ‘printf’
exp.y:19:3: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
exp.y:20:3: warning: implicit declaration of function ‘exit’ [-Wimplicit-function-declaration]
   exit(0);
   ^~~~
exp.y:20:3: warning: incompatible implicit declaration of built-in function ‘exit’
exp.y:20:3: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
exp.y: At top level:
exp.y:23:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
y.tab.c: In function ‘yyparse’:
y.tab.c:45:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]
 # define YYLEX yylex()
                ^
y.tab.c:311:18: note: in expansion of macro ‘YYLEX’
         yychar = YYLEX;
                  ^~~~~
exp.y:6:3: warning: incompatible implicit declaration of built-in function ‘printf’
 stmt: E  NL {printf("valid\n");exit(0);}
   ^~~~~~
exp.y:6:3: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
[root@localhost student]# ./a.out
3+2
valid
[root@localhost student]# ./a.out
3=2
invalid
 syntax error[root@localhost student]# █
```

RESULT: