Ex No: 5
Date:

RECOGNIZE AN ARITHMETIC EXPRESSION USING LEX AND YACC AIM:

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

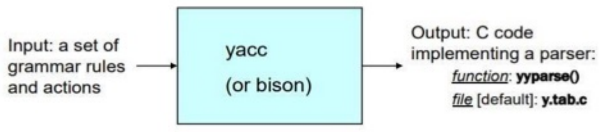
ALGORITHM:

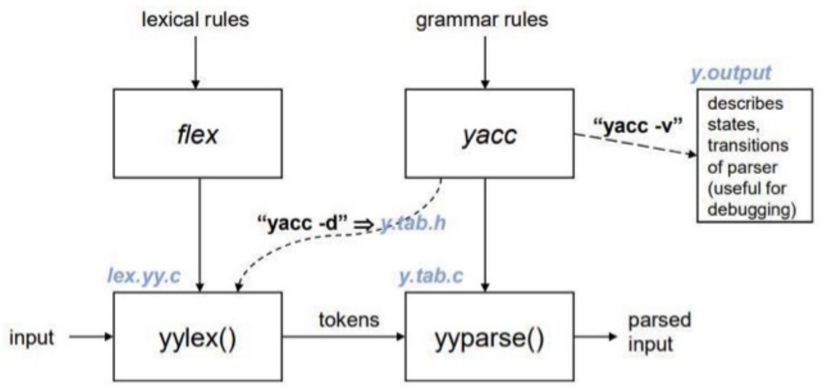
- Using the flex tool, create lex and yacc files.
- In the C include section define the header files required.
- In the rules section define the REGEX expressions along with proper definitions.
- In the user defined section define yywrap() function.
- Declare the yacc file inside it in the C definitions 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;
. return yytext[0];
%%
int yywrap()
{ return
1;
}
```

validexp.y:

```
 %{
   #include<stdio.h>
%}
%token NUMBER
%token VARIABLE

%left '+' '-'
```

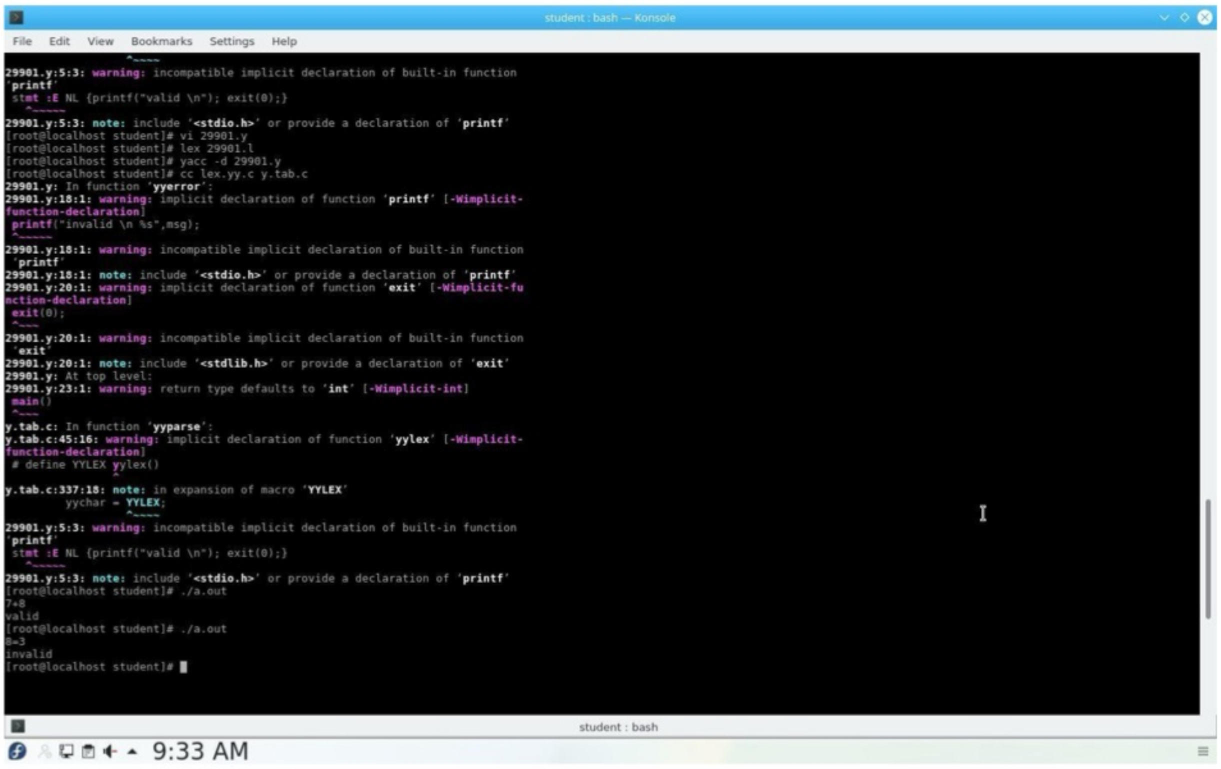
```
%left '*' '/'

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

OUTPUT:



```
                  ^~~~~
29901.y:5:3: warning: incompatible implicit declaration of built-in function
'printf'
 stmt :E NL {printf("valid \n"); exit(0);}
   ^~~~~~
29901.y:5:3: note: include '<stdio.h>' or provide a declaration of 'printf'
[root@localhost student]# vi 29901.y
[root@localhost student]# lex 29901.l
[root@localhost student]# yacc -d 29901.y
[root@localhost student]# cc lex.yy.c y.tab.c
29901.y: In function 'yyerror':
29901.y:18:1: warning: implicit declaration of function 'printf' [-Wimplicit-
function-declaration]
 printf("invalid \n %s",msg);
 ^~~~~~
29901.y:18:1: warning: incompatible implicit declaration of built-in function
 'printf'
29901.y:18:1: note: include '<stdio.h>' or provide a declaration of 'printf'
29901.y:20:1: warning: implicit declaration of function 'exit' [-Wimplicit-fu
nction-declaration]
 exit(0);
 ^~~~
29901.y:20:1: warning: incompatible implicit declaration of built-in function
 'exit'
29901.y:20:1: note: include '<stdlib.h>' or provide a declaration of 'exit'
29901.y: At top level:
29901.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:337:18: note: in expansion of macro 'YYLEX'
         yychar = YYLEX;
                  ^~~~~
29901.y:5:3: warning: incompatible implicit declaration of built-in function
'printf'
 stmt :E NL {printf("valid \n"); exit(0);}
   ^~~~~~
29901.y:5:3: note: include '<stdio.h>' or provide a declaration of 'printf'
[root@localhost student]# ./a.out
7+8
valid
[root@localhost student]# ./a.out
8=3
invalid
[root@localhost student]# 
```

RESULT: