**Ex No:4a YACC – RECOGNISE VALID ARTHIMETIC EXPRESSION OR NOT**

**ALGORITHM:**

**Step1:**Start the program.

**Step2:**Reading an expression .

**Step3:** Checking the validating of the given expression according to the rule using yacc.

**Step4:**Using expression rule print the result of the given values

**Step5:** Stop the program.

**vi 4a.l**

%{

#include"y.tab.h"

#include<math.h>

%}

%%

[a-zA-Z]+ {return id;}

[0-9]+ {return dig;}

[+] {return '+';}

[-] {return '-';}

[\*] {return '\*';}

[/] {return '/';}

[\t]+;

[\n] {return 0;}

. {return yytext[0];}

%%

**vi 4a.y**

%{

#include<stdio.h>

#include<ctype.h>

%}

%token dig

%token id

%left '+' '-'

%left '\*' '/'

%%

st:exp

;

exp:exp '+' exp

|exp '-' exp

|exp '\*' exp

|exp '/' exp

|'{'exp'}'

|dig

|id

;

%%

int main()

{

printf("Enter the expression:");

if(yyparse()==0)

printf("\n Valid\n");

return 0;

}

int yyerror()

{

printf("\n Invalid\n");

return 0;

}

yywrap()

{

}

**OUTPUT**

[cseb100@localhost cseb100]$ lex 4a.l

[cseb100@localhost cseb100]$ yacc -d 4a.y

[cseb100@localhost cseb100]$ gcc lex.yy.c y.tab.c

[cseb100@localhost cseb100]$ ./a.out

Enter the expression:a+b\*c

Valid

[cseb100@localhost cseb100]$ ./a.out

Enter the expression:a+

Invalid

[cseb100@localhost cseb100]$ ./a.out

Enter the expression:1+3

Valid

**Ex No:4b YACC – RECOGNISE VALID IDENTIFIER OR NOT WHICH START WITH A LETTER FOLLOWED BY ANY NUMBER OF LETTER OR DIGITS**

**AIM :**

 To write a yacc program to check valid variable followed by letter or digits

**ALGORITHM:**

**Step1:** Start the program

**Step2:** Reading an expression

**Step3:** Checking the validating of the given expression according to the rule using yacc.

**Step4:** Using expression rule print the result of the given values

**Step5:** Stop the program

**vi 4b.l**

%{

#include "y.tab.h"

%}

%%

([a-zA-z][a-zA-Z0-9]\*) {return ID; }[\n] {return 0;}{return yytext[0];}

%%

**vi 4b.y**

%{

#include<stdio.h>

%}

%token ID

%%

st:ID

%%

int main()

{ printf("\n Enter the string:");

if(yyparse()==0)

printf("\n Valid\n");

return 0;

}

int yyerror()

{ printf("\n Invalid\n");

return 0;

}

yywrap()

{}

**OUTPUT**

[cseb100@localhost cseb100]$ lex 4b.l

[cseb100@localhost cseb100]$ yacc -d 4b.y

[cseb100@localhost cseb100]$ gcc lex.yy.c y.tab.c

[cseb100@localhost cseb100]$ ./a.out

Enter the string:length09

Valid

[cseb100@localhost cseb100]$ ./a.out

Enter the string:12name

Invalid

**Ex No:4c CALCULATOR USING LEX AND YACC**

**ALGORITHM:**

**LEX**

1. Include necessary header files.
2. Define the rule for alphabet and return LETTER.
3. Define the rule for digit and return DIGIT.

If neither alphabet nor digit, return c

**YACC:**

Step1: A Yacc source program has three parts as follows:

       Declarations %%  translation rules %%  supporting C routines

Step2: Declarations Section: This section contains entries that:

 i. Include standard I/O header file.

 ii. Define global variables.

 iii. Define the list rule as the place to start processing.

 iv. Define the tokens used by the parser. v. Define the operators and their precedence.

Step3:  Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

Step4:  Programs Section: The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

Step5:  Main- The required main program that calls the yyparse subroutine to start the program.

Step6:  yyerror(s) -This error-handling subroutine only prints a syntax error message.

Step7:  yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

Step8: calc.lex contains the rules to generate these tokens from the input stream.

**vi 4c.l**

%{ #include"y.tab.h"

#include<math.h>

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext);return dig;}

[+] {return '+';}

[-] {return '-';}

[\*] {return '\*';}

[/] {return '/';}

[\t]+;

[\n] {return 0;}

. {return yytext[0];}

%%

**vi 4c.y**

%{#include<stdio.h>

#include<ctype.h>

%}

%token dig

%left '+' '-'

%left '\*' '/'

%%

st:exp{printf("\nValid:%d",$1);}

;

exp:exp '+' exp {$$=$1+$3;}

|exp '-' exp {$$=$1-$3;}

|exp '\*' exp { $$=$1\*$3;}

|exp '/' exp {if($3==0) yyerror("error"); else $$=$1/$3;}

|dig {$$=$1;}

|'('exp')' {$$=$2;}

;

%%

int main()

{

printf("Enter the expression:");

if(yyparse()==0)

printf("\nSuccess\n");

return 0;

}

int yyerror()

{

printf("\nInvalid\n");

return 0;

}

yywrap()

{

}

**OUTPUT**

[cseb100@localhost cseb100]$ lex 4c.l

[cseb100@localhost cseb100]$ yacc -d 4c.y

[cseb100@localhost cseb100]$ gcc lex.yy.c y.tab.c

[cseb100@localhost cseb100]$ ./a.out

Enter the expression:3+6

Valid:9

Success

[cseb100@localhost cseb100]$ ./a.out

Enter the expression:a+b

Invalid