**(1) Program to calculate total no of vowels followed by another vowel and consonants followed by another consonants and numbers in a given string.**

**// lex vowel.l**

**// cc lex.yy.c -lfl**

**// ./a.out**

%{

int vow\_count=0;

int const\_count =0;

%}

%%

[aeiouAEIOU] {vow\_count++;}

[a-zA-Z] {const\_count++;}

%%

int yywrap(){}

int main()

{

printf("Enter the string of vowels and consonants:");

yylex();

printf("Number of vowels are: %d\n", vow\_count);

printf("Number of consonants are: %d\n", const\_count);

return 0;

}

**(2) Program to find the no. of +ve and -ve integers between -1000 to 1000 and +ve and -ve fraction between -1000 to 1000.**

lex count.l

cc lex.yy.c -o count -ll

%{

#include <stdio.h>

int pos\_int = 0, neg\_int = 0, pos\_frac = 0, neg\_frac = 0;

%}

%%

^-?[0-9]+$ {

int num = atoi(yytext);

if (num > 0) {

pos\_int++;

} else if (num < 0) {

neg\_int++;

}

}

^-?[0-9]+\.[0-9]+$ {

float num = atof(yytext);

if (num > 0) {

pos\_frac++;

} else if (num < 0) {

neg\_frac++;

}

}

%%

int main() {

yylex();

printf("Number of positive integers: %d\n", pos\_int);

printf("Number of negative integers: %d\n", neg\_int);

printf("Number of positive fractions: %d\n", pos\_frac);

printf("Number of negative fractions: %d\n", neg\_frac);

return 0;

}

**(3) Program to recognize valid arithmetic expression using yacc and lex.**

**// lex arxp.l**

**// cc lex.yy.c -lfl**

**// ./a.out**

/\* Lex program to recognize valid arithmetic expression

and identify the identifiers and operators \*/

%{

#include <stdio.h>

#include <string.h>

int operators\_count = 0, operands\_count = 0, valid = 1, top = -1, l = 0, j = 0;

char operands[10][10], operators[10][10], stack[100];

%}

%%

"(" {

top++;

stack[top] = '(';

}

"{" {

top++;

stack[top] = '{';

}

"[" {

top++;

stack[top] = '[';

}

")" {

if (stack[top] != '(') {

valid = 0;

}

else if(operands\_count>0 && (operands\_count-operators\_count)!=1){

valid=0;

}

else{

top--;

operands\_count=1;

operators\_count=0;

}

}

"}" {

if (stack[top] != '{') {

valid = 0;

}

else if(operands\_count>0 && (operands\_count-operators\_count)!=1){

valid=0;

}

else{

top--;

operands\_count=1;

operators\_count=0;

}

}

"]" {

if (stack[top] != '[') {

valid = 0;

}

else if(operands\_count>0 && (operands\_count-operators\_count)!=1){

valid=0;

}

else{

top--;

operands\_count=1;

operators\_count=0;

}

}

"+"|"-"|"\*"|"/" {

operators\_count++;

strcpy(operators[l], yytext);

l++;

}

[0-9]+|[a-zA-Z][a-zA-Z0-9\_]\* {

operands\_count++;

strcpy(operands[j], yytext);

j++;

}

%%

int yywrap()

{

return 1;

}

int main()

{

int k;

printf("Enter the arithmetic expression: ");

yylex();

if (valid == 1 && top == -1) {

printf("\nValid Expression\n");

}

else

printf("\nInvalid Expression\n");

return 0;

}

**(ii)** %{

#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;

}

**(iii)** %{

#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");

}

**yacc -d sample.y**

**lex sample.l**

**gcc lex.yy.c y.tab.c**

**./a.out**

**(4) Program to check balance bracket in an expression using yacc and lex.**

**yacc**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%token LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET

%token END

%%

input:

| input expression END

;

expression:

| LPAREN expression RPAREN

| LBRACE expression RBRACE

| LBRACKET expression RBRACKET

;

%%

int main() {

yyparse();

return 0;

}

int yyerror(const char \*s) {

fprintf(stderr, "%s\n", s);

return 0;

}

**Lex**

%{

#include "y.tab.h"

%}

%%

"(" { return LPAREN; }

")" { return RPAREN; }

"{" { return LBRACE; }

"}" { return RBRACE; }

"[" { return LBRACKET; }

"]" { return RBRACKET; }

\n { return END; }

. { yyerror("invalid character"); }

%%

int yywrap() {

return 1;

}

//////////////////////////////////////////////////////////////////////////

lex -o lex.yy.c balance.l

yacc -d balance.y

gcc lex.yy.c y.tab.c -o balance

./balance