**YACC PROGRAMS**

***Steps to execute YACC program:***

*Step1: vi filename.y (yacc file creation)*

*Step 2: yacc –d filename.y (compiling with yacc –d is optional)*

*Step2: vi filename.l (same name should be used to create its lex part)*

*Step3: cc y.tab.c lex.yy.c –ll*

*Step4:./a.out*

**1. Program to test the validity of a simple expression involving operators**

**+, -, \* and /**

**Yacc Part**

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Valid Expression”); exit(0);}

;

exp : exp ‘+’ exp

| exp ‘-‘ exp

| exp ‘\*’ exp

| exp ‘/’ exp

| ‘(‘ exp ‘)’

| ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the expression\n”);

yyparse();

}

**Lex Part**

%{

#include “y.tab.h”

%}

%%

[0-9]+ { return DIGIT; }

[a-zA-Z][a-zA-Z0-9\_]\* { return ID; }

\n { return NL ;}

. { return yytext[0]; }

%%

**2. Program to recognize nested IF control statements and display the**

**levels of nesting.**

**Yacc Part**

**YACC PART:-**

%{

#include<stdio.h>

#include<stdlib.h>

int count=0;

%}

%token IF RELOP S NUMBER ID NL

%%

stmt: if\_stmt NL {printf("No. of nested if statements=%d\n",count);exit(0);}

;

if\_stmt : IF'('cond')''{'if\_stmt'}' {count++;}

|S

;

cond: x RELOP x

;

x:ID | NUMBER

;

%%

intyyerror(char \*msg)

{

printf("the statement is invalid\n");

exit(0);

}

main()

{

printf("enter the statement\n");

yyparse();

}

**LEX PART:-**

%{

#include "y.tab.h"

%}

%%

"if" {return IF;}

[sS][0-9]\* {return S;}

"<"|">"|"=="|"<="|">="|"!=" {return RELOP;}

[0-9]+ {return NUMBER;}

[a-z][a-zA-Z0-9\_]\* {return ID;}

\n {return NL;}

. {return yytext[0];}

%%

**3. Program to check the syntax of a simple expression involving operators**

**+, -, \* and /**

**Yacc Part**

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Valid Expression”); exit(0);}

;

exp : exp ‘+’ exp

| exp ‘-‘ exp

| exp ‘\*’ exp

| exp ‘/’ exp

| ‘(‘ exp ‘)’

| ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the expression\n”);

yyparse();

}

**Lex Part**

%{

#include “y.tab.h”

%}

%%

[0-9]+ { return NUMBER; }

[a-zA-Z][a-zA-Z0-9\_]\* { return ID; }

\n { return NL ;}

. { return yytext[0]; }

%%

**4. Program to recognize a valid variable, which starts with a letter,**

**followed by any number of letters or digits**.

**Yacc Part**

%token DIGIT LETTER NL UND

%%

stmt : variable NL { printf(“Valid Identifiers\n”); exit(0);}

;

variable : LETTER alphanumeric

;

alphanumeric: LETTER alphanumeric

| DIGIT alphanumeric

| UND alphanumeric

| LETTER

| DIGIT

| UND

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the variable name\n”);

yyparse();

}

**Lex Part**

%{

#include “y.tab.h”

%}

%%

[a-zA-Z] { return LETTER ;}

[0-9] { return DIGIT ; }

[\n] { return NL ;}

[\_] { return UND; }

. { return yytext[0]; }

%%

**5. Program to evaluate an arithmetic expression involving operating +, -,**

**\* and /.**

**Yacc Part**

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Value = %d\n”,$1); exit(0);}

;

exp : exp ‘+’ exp { $$=$1+$3; }

| exp ‘-‘ exp { $$=$1-$3; }

| exp ‘\*’ exp { $$=$1\*$3; }

| exp ‘/’ exp { if($3==0)

{

printf(“Cannot divide by 0”);

exit(0);

}

else

$$=$1/$3;

}

| ‘(‘ exp ‘)’ { $$=$2; }

| ID { $$=$1; }

| NUMBER { $$=$1; }

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the expression\n”);

yyparse();

}

**Lex Part**

%{

#include “y.tab.h”

extern int yylval;

%}

%%

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

\n { return NL ;}

. { return yytext[0]; }

%%

**6. Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using grammar(anbn, n>=0)**

**Yacc Part**

%token A B NL

%%

stmt : s NL { printf(“Valid String\n”); exit(0) ;}

;

s : A s B

|

;

%%

int yyerror(char \*msg)

{

printf(“Invalid String\n”);

exit(0);

}

main ()

{

printf(“Enter the String\n”);

yyparse();

}

**Lex Part**

%{

#include “y.tab.h”

%}

%%

[aA] { return A; }

[bB] { return B; }

\n { return NL ;}

. { return yytext[0]; }

%%

**7. Program to recognize the grammar (anb, n>=10)**

%token A B NL

%%

stmt : A AAAAAAAAA s B NL

{

Printf(“Valid”); exit(0);

}

;

s : s A

|

;

int yyerror(char \*msg)

{

printf(“Invalid String\n”);

exit(0);

}

main ()

{

printf(“Enter the String\n”);

yyparse();

}

**Lex Part**

%{

#include “y.tab.h”

%}

%%

[aA] { return A; }

[bB] { return B; }

\n { return NL ;}

. { return yytext[0]; }

%%