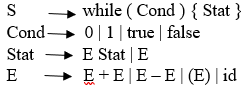
**Experiment No.: 6 Date: 9/11/2020**

**YACC and LEX**

**Aim:** To study YACC and programs that combine YACC and LEX and validate the syntax of:

1. while loop with more than one assignment statements inside while loop.
2. The following set of rules defined in CFG.

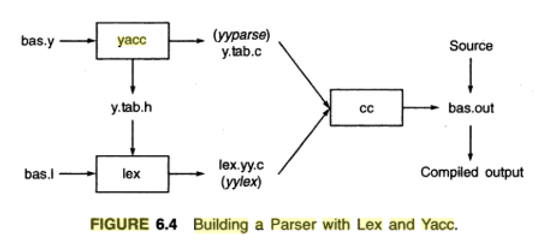


1. Nested if statement with two or more assignment statements.
2. For loop with more than one assignment statement inside for loop

**Theory:**

Lex was designed to create lexical analyzers that could be used with YACC. The LEX library provides a driver program named yylex(). yylex() is required by YACC for its lexical analyzer. If LEX is used to produce the lexical analyzer, we replace the routine yylex() in the third part of the YACC specification by the statement #include “lex.yy.c” and we have each LEX action return a terminal known to YACC.

By using the #include “lex.yy.c” statement, the program yylex has access to YACC’s names for tokens, since the LEX output file is compiled as part of the YACC output file y.tab.c



**References:**

1. Vinu V. Das ; Compiler design with FLEX and YACC; PHI publication, ISBN:978- 81-203-3251-5
2. Aho, Ulman and Sethi; Compilers, Principles, techniques and tools; Pearson Education Asia, ISBN: 81-7808-046-X.

**PROGRAM 1:**

**Yacc prgm:**

%{

#include<stdio.h>

%}

%token MOR NEWLINE

%%

msg: MOR NEWLINE {

char name[20];

printf("MAy I know your name: ");

scanf("%s",name);

printf("\nMorning %s",name);

printf("\n End of successful execution\n");

return;

};

%%

int main()

{

yyparse();

return 0;

}

void yyerror()

{

printf("\nError");

printf("Enter Mornoing only");

}

**Lex prgm:**

%{

#include "y.tab.h"

%}

%%

"Morning" {return MOR;}

[\n] {return NEWLINE;}

[ ] {}

%%

yywrap(){}

**Output:**

root@kali:~# ./a.out

morning

morning

root@kali:~# ./a.out

Morning

MAy I know your name: Praj

Morning Praj

End of successful execution

**PROGRAM 2:**

**Lex prgm:**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[+-] {return \*yytext;}

[\n] {return NL;}

[ ] { }

**Yacc prgm:**

%{

#include<stdio.h>

%}

%token DIGIT NL

%left '+' '-'

%%

E : E '+' T NL {$$=$1+$3; printf("%d + %d = %d ", $1,$3,$$); return;}

| T

;

T : T '-' F NL {$$ = $1 - $3; printf("%d - %d = %d", $1,$3,$$);return;}

|F

;

F : '(' E ')' {$$ = $2;printf(" E = %d", $$);return;}

|DIGIT

;

%%

yywrap(){}

int main(void)

{

printf("\nEnter the expression:");

yyparse();

}

void yyerror()

{

printf("\n ERROR");

}

**Output:**

root@kali:~# ./a.out

Enter the expression:5+5

5 + 5 = 10

root@kali:~# ./a.out

Enter the expression:8\*9

\*

root@kali:~# ./a.out

Enter the expression:9+1

9 + 1 = 10

**PROGRAM 3:**

**Yacc prgm**

%{

#include<stdio.h>

%}

%token NUMBER ID FOR RELOP BRAC DELIM PAREN ASIGN\_OP AUTOINC

%%

stmts : wstmt {printf("\n stmts -> wstmt \n"); return;}

|DELIM {printf("\n stmts -> ; \n");} |astmt stmts {printf("\n stmts -> astmt stmts; \n");} |PAREN stmts PAREN {printf("\n stmts -> {[stmts]\*} \n");}

| {printf("\n stmts -> empty; \n");} ;

wstmt : FOR BRAC astmt expr DELIM expr BRAC stmts {printf("\n wstmt -> for (expr;expr;expr) \n");}

;

expr : expr RELOP expr {printf("\n expr -> expr RELOP expr \n"); }

| expr AUTOINC {printf("\n expr -> expr AUTOINC \n"); }

| NUMBER {printf("\n expr -> NUMBER \n"); }

| ID {printf("\n expr -> ID \n"); }

;

astmt : ID ASIGN\_OP expr DELIM {printf("\n astmt -> ID ASIGN\_OP expr \n"); }

;

%%

**Lex prgm:**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

for {return FOR;}

[(|)] {return BRAC;}

[{|}] {return PAREN;}

[;|,] {return DELIM;}

[a-zA-Z\_][a-zA-Z\_0-9]\* {return ID;}

"=" {return ASIGN\_OP;}

["<"|">"] {return RELOP;}

"<=" {return RELOP;}

">=" {return RELOP;}

"==" {return RELOP;}

[0-9]+ {return NUMBER;}

"++" {return AUTOINC;}

%%

**Output:**

root@kali:~# ./a.out

Enter the statement:for(j=0;i<n;j++)

expr -> NUMBER

astmt -> ID ASIGN\_OP expr

expr -> ID

expr -> ID

expr -> expr RELOP expr

expr -> ID

expr -> expr AUTOINC

**PROGRAM 4**

**lex prgm:**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

if {return IF;}

[(|)] {return BRAC;}

[{|}] {return PAREN;}

[;|,] {return DELIM;}

[a-zA-Z\_][a-zA-Z\_0-9]\* {return ID;}

"=" {return ASIGN\_OP;}

["<"|">"] {return RELOP;}

"<=" {return RELOP;}

">=" {return RELOP;}

"==" {return RELOP;}

[0-9]+ {return NUMBER;}

%%

~

**yacc prgm:**

%{

#include<stdio.h>

%}

%token NUMBER ID IF RELOP BRAC DELIM PAREN ASIGN\_OP

%%

stmts : wstmt {printf("\n stmts -> wstmt \n"); return;}

|DELIM {printf("\n stmts -> ; \n");}

|astmt stmts {printf("\n stmts -> astmt stmts; \n");return;}

| {printf("\n stmts -> empty; \n");}

;

wstmt : IF BRAC expr BRAC stmts {printf("\n stmt -> if (expr); \n");}

| IF BRAC expr BRAC PAREN stmts PAREN {printf("\n stmt -> if (expr){[stmts]\*} \n");}

;

expr : expr RELOP expr {printf("\n expr -> expr RELOP expr \n"); }

| NUMBER {printf("\n expr -> NUMBER \n"); }

| ID {printf("\n expr -> ID \n"); }

;

astmt : ID ASIGN\_OP expr DELIM {printf("\n astmt -> ID ASIGN\_OP expr \n"); }

;

%%

int yywrap(){return 1;}

void yyerror(char \*s)

{

printf("\n ERROR");

}

int main(void)

{

printf("\nEnter the statement:");

yyparse();

}

**Output:**

root@kali:~# ./a.out

Enter the statement:if(true)

expr -> ID

if(true) { if(true) { } else { } }

expr -> ID

expr -> ID

stmts -> empty;

stmt -> if (expr){[stmts]\*}

stmts -> wstmt

**PROGRAM 5:**

**lex prgm**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

while {return WHILE;}

[(|)] {return BRAC;}

[{|}] {return PAREN;}

[;|,] {return DELIM;}

[a-zA-Z\_][a-zA-Z\_0-9]\* {return ID;}

"=" {return ASIGN\_OP;}

["<"|">"] {return RELOP;}

"<=" {return RELOP;}

">=" {return RELOP;}

"==" {return RELOP;}

[0-9]+ {return NUMBER;}

%%

**YACC prgm:**

%{ //while loop with more than one assignment inside while loop expt6\_while\_assgn.y

#include<stdio.h>

%}

%token NUMBER ID WHILE RELOP BRAC DELIM PAREN ASIGN\_OP

%%

stmts : wstmt {printf("\n stmts -> wstmt \n"); return;}

|DELIM {printf("\n stmts -> ; \n");}

|astmt stmts {printf("\n stmts -> astmt stmts; \n");}

| {printf("\n stmts -> empty; \n");}

;

wstmt : WHILE BRAC expr BRAC stmts {printf("\n stmt -> while (expr); \n");}

| WHILE BRAC expr BRAC PAREN stmts PAREN {printf("\n stmt -> while (expr){[stmts]\*} \n");}

;

expr : expr RELOP expr {printf("\n expr -> expr RELOP expr \n"); }

| NUMBER {printf("\n expr -> NUMBER \n"); }

| ID {printf("\n expr -> ID \n"); }

;

astmt : ID ASIGN\_OP expr DELIM {printf("\n astmt -> ID ASIGN\_OP expr \n"); }

;

%%

int yywrap(){return 1;}

void yyerror(char \*s)

{

printf("\n ERROR");

}

int main(void)

{

printf("\nEnter the statement:");

yyparse();

}

**Output:**

root@kali:~# ./a.out

Enter the statement:while(true) { }

expr -> ID

stmts -> empty;

stmt -> while (expr){[stmts]\*}

stmts -> wstmt

**Conclusion:** The Lex and Yacc programs were implemented successfully.