**Commands to execute**

**For both LEX and YACC** lex program1.l

yacc -d program1.y

gcc lex.yy.c y.tab.c -ll

./a.out

**For LEX**

lex prog.l

gcc lex.yy.c

./a.out

**Program 1: a) Write a LEX program to count number of words, lines, characters and whitespaces in a given paragraph.**

%{

#include<stdio.h>

int lines=1, spaces=0, words=0, characters=0;

%}

%%

[ ] {spaces++;}

[\t] {spaces=spaces+3;}

[\n] {lines++;}

[a-zA-Z]\* {words++;characters=characters+yyleng;}

# {return 0;}

%%

int yywrap(){

return 1;

}

int main(){

printf("enter a para\n");

yylex();

printf("Number of lines =%d, number of spaces =%d, number of words =%d,number of characters =%d",lines,spaces,words,characters);

}

*Use # to end the input.*

**Program 1: b) Write a YACC program to recognize strings of the form anbn+mcm, n,m>=0**

**Lex code**

%{

#include "y.tab.h"

%}

%%

'a' {return 'a';}

'b' {return 'b';}

'c' {return 'c';}

. {return yytext[0];} //changed from . to .{return yytext[0];}

[\n] {return 0;}

%%

**YACC code**

%{

#include<stdio.h>

#include<string.h>

void yyerror(char const\*s);

%}

%start S

%%

S: A B ;

A: 'a'A'b' | ;

B: 'b'B'c' | ;

%%

int main(){

printf("Enter words\n");

yyparse();

printf("true\n");

return 0;

}

void yyerror(char const \*s){

fprintf(stderr,"Invlaid\n");

exit(0);

}

**Program 2: a) Write a LEX program to count number of Positive and Negative integers and Positive & Negative fractions.**

%{

int nf=0,p=0,n=0,pf=0;

%}

%%

[+]?[0-9]\* {p++;}

[-][0-9]\* {n++;}

[+]?[0-9]\*[.][0-9]\* {pf++;}

[-][0-9]\*[.][0-9]\* {nf++;}

[+]?[0-9]\*[.]\*[0-9]\*[/][+]?[0-9]\*[.]\*[0-9]\* {pf++;}

[+]?[0-9]\*[.]\*[0-9]\*[/][-][0-9]\*[.]\*[0-9]\* {nf++;}

[-][0-9]\*[.]\*[0-9]\*[/][+]?[0-9]\*[.]\*[0-9]\* {nf++;}

[-]?[0-9]\*[.]\*[0-9]\*[/][-]?[0-9]\*[.]\*[0-9]\* {pf++;}

# {return 0;}

%%

int yywrap(){

return 1;

}

int main(){

printf("Enter numbers, (use # to end )\n");

yylex();

printf("Number of positive integer is= %d\n,Number of negetive integer is= %d\n,Number of positive fractions is= %d\n,Number of negetive fractions is= %d\n",p,n,pf,nf);

}

**Program 2: b) Write a YACC program to validate and evaluate a simple expression involving operators +,- , \* and /**

**LEX File**

%{

#include "y.tab.h"

extern YYSTYPE yylval;

%}

%%

[0-9]\* {yylval=atoi(yytext);return NUM;}

[-+\*/] {return yytext[0];}

. {return yytext[0];};

\n {return 0;}

%%

**YACC File**

%{

#include<stdio.h>

#include<stdlib.h>

void yyerror();

int yylex(void);

%}

%token NUM;

%left '+' '-'

%left '/' '\*'

%%

S : I {printf("Result is %d\n",$$); };

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

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

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

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

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

| NUM {$$=$1;}

| '-'NUM {$$=-$2;}

;

%%

int main(){

printf("Enter an expression\n");

yyparse();

printf("Valid\n");

return 0;

}

void yyerror(){

printf("Invlaid\n");

exit(0);

}

Note: Use gcc **lex.yy.c y.tab.c -ll** and not **gcc lex.yy.c y.tab.h -ll**

**Program 3a: Write a LEX program to count the number of comment lines in a C Program. Also eliminate them and copy that program into a separate file.**

%{

#include<stdio.h>

#include<stdlib.h>

int single=0,multi=0,multilines=0;

%}

%%

"//"[^\n]\* {single++;}

"/\*"[^\*]\*"\*/" {

multi++;

for(int i=0;i<yyleng;i++){

if(yytext[i]=='\n')

multilines++;

}

}

%%

int yywrap(){

return 1;

}

int main(){

yyin=fopen("inp.txt","r");

yyout=fopen("out.txt","w");

yylex();

printf("Single lines=%d\nMultiline comment=%d\nNumber of lines in the multiline comment=%d\n",single,multi,multilines);

return 1;

}

**Inp.txt**

int main(){

//This is a sinle line comment

printf("Welcome to Hell\n");

/\*This

Is a

Multi line

Comment

\*/

int a,b;

}

**Program 3b: Write a YACC program to recognize a nested (minimum3levels)FOR loop statement for C language.**

**LEX code**

%{

#include "y.tab.h"

%}

%%

"for" { return FOR; }

"(" { return LPAREN; }

")" { return RPAREN; }

"{" { return LF; }

"}" { return RF; }

"=" { return '='; }

"-" { return '-'; }

"+" { return '+'; }

">" { return '>'; }

"<" { return '<'; }

";" { return ';'; }

"==" { return EQ; }

"<=" { return LE; }

">=" { return GE; }

"+=" { return ADDEQ; }

"-=" { return SUBEQ; }

"++" { return INC; }

"--" { return DEC; }

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

[0-9]+ { return NUM; }

[ \t] { /\* Ignore \*/ }

# { return 0; }

. { /\* Ignore \*/ }

%%

int yywrap(){

return 1;

}

**Input**

for(i=0;i<7;i++){

for(j=0;j<9;j++){

}

}

**Invalid inputs**

for(int i=0;i<7;i++){

for(int j=0;j<9;j++){

}

}

for(i=0;i<7;i++){

int a,b;

for(j=0;j<9;j++){

}

}

**YACC code**

%{

#include <stdio.h>

#include <stdlib.h>

int yylex();

int count = 0;

int error=0;

int yyerror();

%}

%token FOR LPAREN RPAREN LF RF ALPH NUM EQ LE GE ADDEQ SUBEQ INC DEC

%%

S : I

;

I : FOR A B { count++; }

;

A : LPAREN E ';' E ';' E RPAREN

;

E : ALPH Z NUM

| ALPH Z ALPH

| ALPH U

| /\* empty \*/

;

Z : '='

| '>'

| '<'

| LE /\* Placeholder for '<=' \*/

| GE /\* Placeholder for '>=' \*/

| EQ /\* Placeholder for '==' \*/

| ADDEQ /\* Placeholder for '+=' \*/

| SUBEQ /\* Placeholder for '-=' \*/

;

U : INC /\* Placeholder for '++' \*/

| DEC /\* Placeholder for '--' \*/

;

B : LF B RF

| I

| ALPH

| ALPH I

| /\* empty \*/

;

%%

int main() {

yyparse();

if(error){

printf("error");

}

else{

printf("valid");

}

printf("Number of nested FOR's are: %d\n", count);

return 0;

}

int yyerror() {

error=1;

exit(0);

}

**Enter code snippet**

for(i=0;i<5;i++){

for(j=0;j<6;j++){

}

}

#

Valid number of for loops are 2

**Program 4a: Write a LEX program to recognize and count the number of identifiers, operators and keywords in a given input file.**

%{

#include<stdio.h>

int key=0,id=0,op=0;

%}

%%

"int"|"float"|"double"|"if"|"for"|"else"|"while"|"switch"|"printf"|"scanf"|"exit"|"return"|"case"|"main()" {key++;}

[a-zA-Z\_][a-zA-Z0-9\_]\* {id++;}

[0-9]

[\+\-\\*\/\&\|\!\(\)\{\}] {op++;}

[;]

[,]

[\t\n]+

[#] {return 0;}

[.]+ {printf("Invalid%s\n",yytext);}

%%

int yywrap(){

return 1;

}

int main(){

yyin=fopen("eg.c","r");

yylex();

printf("Keywords=%d\nIdentifiers=%d\nOperators=%d\n",key,id,op);

return 0;

}

Eg.c

int main(){

int a,b;

return 0;

}

**Program 4b: Write a YACC program to recognize nested IF control statements (C language) and display the number of levels of nesting.**

**LEX code:**

%{

#include "y.tab.h"

%}

%%

"if" {return IF;}

"(" { return LPAREN; }

")" { return RPAREN; }

"{" { return LF; }

"}" { return RF; }

[;] {return ';';}

[,]

"<" {return '<';}

">" {return '>';}

"=" {return '=';}

"+" {return '+';}

"-" {return '-';}

"==" {return EQ;}

"<=" {return LEQ;}

">=" {return GEQ;}

[ \t\n]\*

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

[0-9]+ {return NUM;}

# {return 0;}

.

%%

**YACC Code**

%{

#include<stdio.h>

#include<stdlib.h>

int count=0;

void yyerror();

int yylex();

%}

%token IF ALPHA NUM GEQ LEQ EQ LPAREN RPAREN LF RF

%%

S:I;

I: IF A B {count++;};

A: LPAREN E RPAREN ;

E: ALPHA Z ALPHA| ALPHA Z NUM| ;

Z: '<'|'>'|GEQ|LEQ|EQ ;

B: ALPHA|ALPHA I|LF B RF| I|;

%%

int main(){

printf("enter an expression\n");

yyparse();

printf("Number of if loops are %d",count);

return 0;

}

void yyerror(){

printf("Invalid\n");

exit(0);

}

**Program 5: Write a YACC program to recognize Declaration statement (C language) and display the number variables declared .**

**Variable can be any basic data type or array type**

**Example int a[10],a,b,c;**

**LEX Code**

%{

#include "y.tab.h"

%}

extern YYSTYPE yylval;

%%

"int" { return INT; }

"float" { return FLOAT; }

"char" { return CHAR; }

"double" { return DOUBLE; }

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

[0-9]+ {return NUM;}

"[" { return '['; }

"]" { return ']'; }

"," { return ','; }

";" { return ';'; }

[ \t\n] { /\* Ignore whitespace \*/ }

# { return 0; }

%%

int yywrap() {

return 1;

}

**YACC Code**

%{

#include <stdio.h>

#include <stdlib.h>

int var\_count = 0;

void yyerror(const char \*s);

int yylex();

%}

%token INT FLOAT CHAR DOUBLE NUM IDENTIFIER

%%

program: declarations

;

declarations: declaration ';'

| declarations declaration ';'

;

declaration: type var\_list

;

type: INT

| FLOAT

| CHAR

| DOUBLE

;

var\_list: var

| var\_list ',' var

;

var: identifier

| identifier '[' ']' // Matches array without size

| identifier '[' NUM ']' // Matches array with size

;

identifier: IDENTIFIER

{

var\_count++;

}

;

%%

void yyerror(const char \*s) {

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

}

int main() {

yyparse();

printf("Total number of variables declared: %d\n", var\_count);

return 0;

}

**Program 7: Write a YACC program that identifies the Function Definition of C language**

**LEX Code**

%{

#include "y.tab.h"

%}

%%

"int"|"void"|"char"|"float"|"double" { return TYP; }

"return" { return RETURN; }

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

"(" { return LP; }

")" { return RP; }

"{" { return LB; }

"}" { return RB; }

";" { return SC; }

"," { return CM; }

"=" { return EQ; }

"+"|"-"|"\*"|"/" { return OP; }

[0-9]+ { return NUM; }

[ \t\n] { /\* ignore whitespace \*/ }

. { /\* ignore other characters \*/ }

%%

int yywrap(void) {

return 1;

}

**YACC Code**

%{

#include <stdio.h>

void yyerror(const char \*s);

%}

%token TYP ID LP RP LB RB SC CM EQ OP RETURN NUM

%left OP

%left EQ

%%

prog: funcs ;

funcs: func | funcs func ;

func: TYP ID LP params RP LB stmts RB {

printf("Function is syntactically correct.\n");

} ;

params: /\* empty \*/ | param\_list;

param\_list: param | param\_list CM param ;

param: TYP ID ;

stmts:stmt | stmts stmt ;

stmt: var\_decl | expr SC | RETURN expr SC ;

var\_decl: TYP ID SC | TYP ID EQ expr SC ;

expr: ID| NUM | ID EQ expr | expr OP expr| LP expr RP ;

%%

void yyerror(const char \*s) {

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

}

int main(void) {

return yyparse();

}

**Input**

int sum(int a,int b){

int ans=a+b;

return ans;

}

**Program 6: YACC program that reads the C statements for an input file and converts them in quadruple three address intermediate code.**

**Lex code**

%{

#include "y.tab.h"

extern char yyval;

%}

%%

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

[a-zA-Z]+ { yylval.sym = (char)yytext[0]; return LETTER; }

\n { return 0; }

. { return yytext[0]; }

%%

int yywrap() { return 1; }

**YACC Code**

%{

#include <stdio.h>

#include <stdlib.h>

struct incod {

char opd1, opd2, opr;

} code[20];

int ind = 0;

int flag = 0;

char temp = 'T'; // Start with 'T'

char AddToTable(char, char, char);

void generateCode();

%}

%union { char sym; }

%token <sym> LETTER NUMBER

%type <sym> expr

%left '-' '+'

%right '\*' '/'

%%

statement: LETTER '=' expr ';' { AddToTable($1, $3, '='); }

| expr ';' ;

expr: expr '+' expr { $$ = AddToTable($1, $3, '+'); }

| expr '-' expr { $$ = AddToTable($1, $3, '-'); }

| expr '\*' expr { $$ = AddToTable($1, $3, '\*'); }

| expr '/' expr { $$ = AddToTable($1, $3, '/'); }

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

| NUMBER { $$ = $1; }

| LETTER { $$ = $1; }

;

%%

char AddToTable(char opd1, char opd2, char opr) {

code[ind++] = (struct incod){ opd1, opd2, opr };

char retTemp = temp;

// Cycle through 'T', 'U', 'V', ... by incrementing the character

if (temp < 'Z') {

temp++; // Increment to next character

}

return retTemp;

}

void generateCode() {

printf("\nThree-Address Code:\n");

for (int i = 0; i < ind; i++){

if(i==ind-1){

printf("%c %c %c\n", code[i].opd1, code[i].opr, code[i].opd2);

break;

}

printf("%c = %c %c %c\n", temp - ind + i, code[i].opd1, code[i].opr, code[i].opd2);

}

printf("\nQuadruple Code:\n");

for (int i = 0; i < ind; i++){

if(i==ind-1){

printf("%d\t%c\t%c\t%c\n", i, code[i].opr, code[i].opd1, code[i].opd2);

break;

}

printf("%d\t%c\t%c\t%c\t%c\n", i, code[i].opr, code[i].opd1, code[i].opd2, temp - ind + i);

}

printf("\nTriple Code:\n");

for (int i = 0; i < ind; i++)

printf("%d\t%c\t%c\t%c\n", i, code[i].opr, code[i].opd1, code[i].opd2);

}

int main() {

printf("Enter the Expression (e.g. a = b + c;): ");

yyparse();

if (flag == 0)

generateCode();

return 0;

}

int yyerror(char \*s) {

flag = 1;

printf("%s\n", s);

return 0;

}

**Program 8: Write a YACC program that generates Assembly language (Target) Code for valid Arithmetic Expression.**

**Lex code**

%{

#include "y.tab.h"

#include <stdlib.h>

#include <string.h>

%}

DIGIT [0-9]

ID [a-zA-Z][a-zA-Z0-9]\*

WS [ \t\n]

STRING \"[^"]\*\"

%%

"int" { return INT; }

"main" { return MAIN; }

"printf" { return PRINTF; }

{STRING} { yylval.str = strdup(yytext); return STRING; }

{ID} { yylval.id = strdup(yytext); return ID; }

{DIGIT}+ { yylval.num = atoi(yytext); return NUM; }

"+" { return ADD; }

"=" { return ASSIGN; }

"(" { return LPAREN; }

")" { return RPAREN; }

";" { return SEMI; }

"," { return COMMA; }

"{" { return LBRACE; }

"}" { return RBRACE; }

{WS} ; /\* ignore whitespace \*/

%%

int yywrap() {

return 1;

}

**YACC Code**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern int yylex();

extern int yylineno;

void yyerror(const char\* s) {

fprintf(stderr, "Error: %s at line %d\n", s, yylineno);

exit(1);

}

%}

%union {

char\* id;

int num;

char\* str;

}

%token <id> ID

%token <num> NUM

%token <str> STRING

%token INT MAIN PRINTF ADD LPAREN RPAREN SEMI COMMA LBRACE RBRACE ASSIGN

%start program

%%

program:

INT MAIN LPAREN RPAREN LBRACE stmt\_list RBRACE

{

printf(".data\n");

printf(" .LC0: .string \"Sum %%d\"\n");

printf(".text\n");

printf(" .globl main\n");

printf("main:\n");

}

;

stmt\_list:

stmt

| stmt\_list stmt

;

stmt:

INT ID ASSIGN NUM SEMI {

printf(" movl $%d, %s\n", $4, $2);

}

| ID ASSIGN ID ADD ID SEMI {

printf(" movl %s, %%eax\n", $3);

printf(" addl %s, %%eax\n", $5);

printf(" movl %%eax, %s\n", $1);

}

| PRINTF LPAREN STRING COMMA ID RPAREN SEMI {

printf(" movl %s, %%edi\n", $5); // Load argument into %edi

printf(" movl $.LC0, %%rsi\n"); // Address of format string into %rsi

printf(" call printf\n"); // Call printf function

}

;

%%

int main() {

printf("Assembly code output:\n");

yyparse();

return 0;

}

lex program1.l

yacc -d program1.y

gcc lex.yy.c y.tab.c -o output -ll

echo '#int main(){int a=5;int b=10; a=a+b; printf("Sum %d\\n",a);}'|./output