Basic Commands

- ### To compile

```
lex [name].l
```

This creates a C file called `lex.yy.c`

- ### To Run this

```
gcc lex.yy.c
```

- This creates a successful file for running it

```
./a.out
```

Basic Structure

```
%{
#include <stdio.h>
//All other declarations that are required
%} //Notice How the Preprocessor are kept
//All the various Regular Expressions are here
%%
//All the various Rules that are defined in the file
%%
int main(){}
```

Experiment 1

To Count the Number Of Lines, tabs, spaces, etc. from a given Text File

```
%{
#include <stdio.h>
int count=0;
%}

%%
[^\t\n]+   {count+=yyleng;}
[] {count++;}
```

```
.
.
%%
int main()
{
yyin = fopen("a1.txt","w");
yylex();
fclose(yyin);
}
int yywrap()
{
printf("The number of lines is:%d",count);
}
```

Experiment 2

Lexical Analyzer for Sample Language using Lex

```
%{
#include <stdio.h>
%}
L   [a-zA-Z]
D   [0-9]
Id  {L}({L}|{D})*
num {D}+(\.{D}+)?([eE][-+]?{D}+)?
bop [-+*/=]
uop "++"|"--"
relop  "<"|"<="|">"|">="|"!="|"=="
lop   "&&"|"||"
kew    "class"|"public"|"void"|"String"|"System"|"static"|"out"|"println"
pun   [,:’”\[\]\{\}\)\(.;]
ws  [\t\n]
st   \".*\"
%%
{ws}  {}
{kew} {printf( "keyword=%s\n", yytext);}
{Id} {printf( "identifier =%s\n", yytext);}
{num}  {printf( "constant =%s\n", yytext);}
{bop} {printf( "binary op=%s\n", yytext);}
{uop} {printf( "unary op=%s\n", yytext);}
{relop} {printf( "relational op=%s\n", yytext);}
{lop} {printf( "logical op=%s\n", yytext);}
{pun} {printf( "punct =%s\n", yytext);}
{bitlop} {printf( "bitwise logical op=%s\n", yytext);}
{comment} {printf( "comment=%s\n", yytext);}
{st} {printf( "string=%s\n", yytext);}
%%
int main()
```

```
{
    yyin=fopen("prog.java","r");
    yylex();
    fclose(yyin);
    return 0;
}
int yywrap()
{
    return 1;
}
```

Experiment 03

Program for syntax checking of subset of given language using LEX and YACC

- .y

```
%{
#include<stdio.h>
%}
%token DET NOUN VERB PRONOUN CONJ PREPOSITION
%nonassoc PREPOSITION
%nonassoc CONJ
%nonassoc NOUN
%{
int isCompound = 0;
%}
%%
S1: S1 S
  | S
  ;
S: SS '.' { printf("Simple Sentence\n"); }
 | CS '.' { if(isCompound) printf("Compound Sentence\n"); else
printf("Simple Sentence\n"); }
 ;
CS: SS CONJ SS { printf("cs\n"); isCompound = 1; }
 | CS CONJ SS { printf("ss\n"); }
 ;
SS: NP VP { printf("ss\n"); }
 | VP { printf("1\n"); }
 ;
NP: DET NOUN { printf("2\n"); }
 | NOUN { printf("3\n"); }
 | PRONOUN { printf("4\n"); }
 ;
VP: VERB NP { printf("5\n"); }
 | VERB { printf("6\n"); }
 | VP PP { printf("7\n"); }
 ;
PP: PREPOSITION NP { printf("8\n"); }
 ;
%%
int main()
{
    yyparse();
    return 0;
}
int yyerror(char *msg)
{
    printf("%s\n", msg);
    return 1;
}
```

LEX File

- .l

```
%{
#include <stdio.h>
#include "y.tab.h" //See how The Yacc file is called
%}

determiner "this"|"that"

Noun "John"|"door"|"cat"|"bird"|"nests"|"boy"|"girl"|"water"|"song"|"good"

Verb "runs"|"eats"|"drinks"|"sings"|"plays"|"swims"

Preposition "from"|"with"|"on"

Pronoun "she"|"he"|"they"|"I"|"you"|"am"

Conjunction ("and"|"or"|"but")

%%
{Conjunction} {return CONJ;}
{determiner} {return DET;}
{Noun} {return NOUN;}
{Verb} {return VERB;}
{Preposition} {return PREPOSITION;}
{Pronoun} {return PRONOUN;}
"."  {return yytext[0];}
%%
int yywrap()
{
    return 1;
}
```

Generate the parser: `yacc -d parser.y`
This Generates:
- `y.tab.c`
- `y.tab.h`

Generate the lexer: `lex scanner.l`
This Generates:
- `lex.yy.c`

Compile & Run: `gcc y.tab.c lex.yy.c -o parser -ll`

Experiment 04

Program for syntax checking of control statements using LEX and YACC.

- ## .l

```
%{
#include <stdio.h>
#include "y.tab.h"
%}

L [A-Za-z]
D [0-9]
id {L}({L}|{D})*

%%
"if"     { return IF; }
"else"   { return ELSE; }
"for"    { return FOR; }
"do"     { return DO; }
"while"  { return WHILE; }
"++"     { return INC; }
"--"     { return DEC; }
"||"     { return OR; }
"&&"     { return AND; }
"!"      { return NOT; }
"switch" { return SWITCH; }
"case"   { return CASE; }
"break"  { return BREAK; }
"default"{ return DEFAULT; }
[0-9]+(\.[0-9]+)?  { return NUM; }
{id}    { return id; }
"<"|"≤"|">"|"≥"|"=="|"≠"  { return relop; }
[-/;=+*,\(\)\{\}:] { return yytext[0]; }
```

```
[ ]     { /* Ignore whitespace */ }
\n      { /* Ignore newlines */ }
%%

int yywrap() {
    return 1;
}
```

- .y

```
%{
#include <stdio.h>
%}

%token id NUM OR AND NOT relop TRUE FALSE INC DEC IF ELSE DO WHILE uminus
FOR SWITCH CASE BREAK DEFAULT

%right '='
%left '+' '-'
%left '*' '/'
%right '^'
%nonassoc uminus
%left OR
%left AND
%nonassoc NOT

%%
S1 : S1 S
   | S
   ;

S : AS ';'      { printf("Assignment statement accepted\n"); }
  | IFS         { printf("If statement is accepted\n"); }
  | IFES        { printf("If else statement is accepted\n"); }
  | WS          { printf("While statement is accepted\n"); }
  | DWS         { printf("Do while statement is accepted\n"); }
  | FORS        { printf("For statement is accepted\n"); }
  | SS          { printf("Switch statement is accepted\n"); }
  ;

SS : SWITCH '(' E ')' '{' CV '}'
   ;

CV : CASE E ':' S1 BREAK ';'
   | CASE E ':' S1 BREAK ';' CV
   | CASE E ':' S1 BREAK ';' DEFAULT ':' S1
   ;
```

```
AS : id '=' E
   ;

E : E '+' E
  | E '-' E
  | E '*' E
  | E '/' E
  | E '^' E
  | '-' E %prec uminus
  | id
  | NUM
  ;

IFS : IF '(' BE ')' '{' S1 '}'
    ;

BE : BE OR BE
   | BE AND BE
   | NOT BE
   | id relop id
   | TRUE
   | FALSE
   ;

IFES : IF '(' BE ')' '{' S1 '}' ELSE '{' S1 '}'
     ;

WS : WHILE '(' BE ')' '{' S1 '}'
   ;

DWS : DO '{' S1 '}' WHILE '(' BE ')' ';'
    ;

FORS : FOR '(' IS ';' BE ';' MS ')' '{' S1 '}'
     ;

IS : AS
   | IS ',' AS
   ;

MS : IS
   | id INC
   | INC id
   | id DEC
   | DEC id
   ;

%%
```

```
void main() {
    yyparse();
}

int yyerror(char *msg) {
    printf("%s\n", msg);
    return 0;
}
```

Experiment 5

Program for Syntax checking of declaration statement using LEX and YACC.

- .y

```
%{
#include <stdio.h>
%}

%token Int Char Float Bool String IntV CharV FloatV BoolV StringV Id Am
Const

%%
S1 : S1 S
   | S
   ;

S : Int Iv ';'    { printf("int declaration accepted\n"); }
  | Char Cc ';'   { printf("char declaration accepted\n"); }
  | Float Ff ';'  { printf("float declaration accepted\n"); }
  | Bool Bb ';'   { printf("bool declaration accepted\n"); }
  ;

Iv : IdM
   | Iv ',' Id
   | Id '=' IntV
   | Iv ',' Id '=' IntV
   ;

Cc : IdM
   | Cc ',' Id
   | Id '=' CharV
   | Cc ',' Id '=' CharV
   ;

Ff : IdM
```

```
    | Ff ',' Id
    | Id '=' FloatV
    | Ff ',' Id '=' FloatV
    ;

Bb : IdM
    | Bb ',' Id
    | Id '=' BoolV
    | Bb ',' Id '=' BoolV
    ;

Ss : IdM
    | Ss ',' Id
    | Id '=' StringV
    | Ss ',' Id '=' StringV
    ;

IdM : Id
     ;

%%
void yyerror(char *s) {
    printf("%s\n", s);
}

int main() {
    yyparse();
    return 0;
}
```

- **.l**

```
%{
#include "y.tab.h"
%}

letter [a-zA-Z]
num [0-9]+
float {num}+\.{num}+
bools "true"|"false"
identifier {letter}({letter}|{num})*

%%
"int"    { return Int; }
"char"   { return Char; }
"float"  { return Float; }
"bool"   { return Bool; }
{num}    { return IntV; }
```

```
{float}  { return FloatV; }
{bools}  { return BoolV; }
{identifier} { return Id; }
"'"(.)"'" { return CharV; }
[,;=] { return yytext[0]; }
%%

int yywrap() {
    return 1;
}
```

Experiment 06

Implement a desk calculator using LEX and YACC.

Calculator.l (LEX file)

```
%{
#include "y.tab.h"
%}

digit [0-9]
%%
[ \t]         ;   // Ignore whitespace
\n            { return '\n'; }
{digit}+      { yylval = atoi(yytext); return NUMBER; }
"+"           { return '+'; }
"-"           { return '-'; }
"*"           { return '*'; }
"/"           { return '/'; }
"("           { return '('; }
")"           { return ')'; }
.             { printf("Invalid character: %s\n", yytext); }
%%

int yywrap() {
    return 1;
}
```

Calculator.y (YACC file)

```
%{
#include <stdio.h>
#include <stdlib.h>

void yyerror(const char *s);
int yylex();
%}

%token NUMBER

%%
lines:
    lines expr '\n'  { printf("Result: %d\n", $2); }
  | /* empty */
  ;

expr:
    expr '+' term    { $$ = $1 + $3; }
  | expr '-' term    { $$ = $1 - $3; }
  | term             { $$ = $1; }
  ;

term:
    term '*' factor  { $$ = $1 * $3; }
  | term '/' factor  { $$ = $1 / $3; }
  | factor           { $$ = $1; }
  ;

factor:
    '(' expr ')'     { $$ = $2; }
  | NUMBER           { $$ = $1; }
  ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    printf("Simple Desk Calculator\n");
    printf("Enter expressions followed by newline. Press Ctrl+D to
exit.\n");
    yyparse();
    return 0;
}
```