COMPILATION AND EXECUTION COMMANDS :-

FOR LEX ONLY PROGRAM :-

```
1) lex filename.l
2) gcc lex.yy.c -ll
3) ./a.out
```

FOR LEX & YACC PROGRAM :-

```
1) lex filename.l
2) yacc -d filename.y
2) gcc lex.yy.c y.tab.c -ll
3) ./a.out
```

Exp 1 - VALID IDENTIFIER

```
digit [0-9]
letter [a-zA-Z]
arithmetic_operator [-\+\*/%]
space [" "\t]
special [\\\'\":~`!@#\$\^\&\(\)\?/\.<>]
keywords ("int"|"float"|"double"|"long"|"for"|"while"|"if"|"else"|"global"|"break"|"continue")
%%
{keywords} printf("\n%s is a keyword\n",yytext);
({letter}|_)(_|{letter}|{digit})* printf("\n%s is a valid identifier\n",yytext);
{digit}+ printf("\n%s is a number\n",yytext);
{arithmetic_operator} printf("\n%s is an arithmetic operator\n",yytext);
= printf("\n%s is an operator\n",yytext);
; printf("\n%s is a token\n",yytext);
, printf("\n%s is a token\n",yytext);
\n printf("\nnew line\n");
{space}+ continue;
(_|{letter}|{digit}|{special})(_|{letter}|{digit}|{special})* printf("\n%s is not a valid identifier\n",yytext);
%%

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

Exp 2 - BRANCHING STATEMENTS IN C

LEX :-

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

%%
if                          return IF;
else                        return ELSE;
"("                         return OP;
")"                         return CP;
"{"                         return OC;
"}"                         return CC;
";"                         return SC;
[0-9]+                      return NUM;
[-+*/><=”<=””>=””==”]       return OPR;
["&&""||"]                  return LG;
(_|[a-zA-Z])([a-zA-Z0-9])*    return ID;
\n                          return 0;
%%
```

YACC :-

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

%token IF ELSE NUM ID OPR LG OP CP OC CC SC
%start stmt

%%
stmt : S                                {printf("\nValid\n");};
S : STMT IF OP COND CP OC A                         ;
COND : V O V COMB COND | V O V      ;
COMB : LG                               ;
V : ID | NUM                            ;
O : OPR                                         ;
A : B | STMT CC ELSE OC B | S A         ;
B : STMT CC STMT                                    ;
STMT : STMT STMT | V O V SC | S |
;
%%

yyerror()
{
printf("\nError\n");
}

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

Exp 3 - LOOPING STATEMENTS IN C

LEX :-

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

%%
int|float|bool                  return DT;
for                             return FOR;
while                           return WHILE;
do                              return DO;
";"                             return SC;
"("                             return OP;
")"                             return CP;
"{"                             return OC;
"}"                             return CC;
[0-9]+                          return NUM;
=                               return EQ;
"<"|">"|"<="|">="|"=="|"!="    return ROP;
[-+*/]                          return AOPR;
"&&"                            return WHLCMB;
"||"                            return WHLCMB;
","                             return FORCMB;
"++"                            return UPDOPR;
"--"                            return UPDOPR;
(_|[a-zA-Z])(_|[a-zA-Z0-9])*    return ID;
\n                              return 0;
%%
```

YACC :-

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

%token FOR WHILE DO SC OP CP OC CC NUM LG ID INOPR LT GT EQ
%start stmt

%%
stmt : S            {printf("\nValid\n");};
S : FLOOP S| WLOOP S| DLOOP S|
;
FLOOP : FOR OP FCONDN CP OC A    ;
FCONDN : INIT SC CHCK SC UPD      ;
INIT : CONDN1 COMB INIT| CONDN1   ;
CHCK : CONDN2 COMB CHCK| CONDN2         ;
```

```
UPD  : ID INOPR COMB UPD| ID INOPR ;
CONDN1 : V EOPR V                ;
CONDN2 : V ROPR V                ;
EOPR : EQ                        ;
ROPR : LT | GT                                  ;
V : ID | NUM                     ;
COMB : LG                        ;
WLOOP : WHILE OP WCONDN CP OC A          ;
WCONDN : CONDN COMB WCONDN | CONDN ;
CONDN : V AOPR V                 ;
AOPR : EQ | LT | GT              ;
DLOOP : DO OC DA                 ;
DA : STMT CC WHILE OP WCONDN CP SC      ;
A : STMT CC | STMT CC FLOOP | STMT CC WLOOP | STMT CC DLOOP ;
STMT : STMT STMT | V AOPR V SC | S |
;
%%

yyerror()
{
printf("\nError\n");
}

void main()
{
yyparse();
} */

%{
#include<stdio.h>
%}

%token FOR WHILE DO SC OP CP OC CC NUM WHLCMB FORCMB ID UPDOPR ROP AOPR EQ DT
%start stmt

%%
stmt : S            {printf("\nValid\n");};
S : STMT FLOOP | STMT WLOOP | STMT DLOOP ;
FLOOP : FOR OP FCONDN CP OC A CC STMT;
WLOOP : WHILE OP WCONDN CP OC A CC STMT;
DLOOP : DO OC A CC WHILE OP WCONDN CP SC STMT;
A : STMT | A STMT | ; /* epsilon production */
FCONDN : INIT SC CHCK SC UPD;
INIT : CONDN1 FORCMB INIT | CONDN1;
CHCK : CONDN2 FORCMB CHCK | CONDN2;
UPD : ID UPDOPR FORCMB UPD | ID UPDOPR;
CONDN1 : V EQ V | DT V EQ V;
CONDN2 : V ROP V;
V : ID | NUM;
```

```
WCONDN : CONDN WHLCMB WCONDN | CONDN;
CONDN : V ROP V;
STMT : STMT STMT | V EQ V SC | V UPDOPR SC | V EQ V AOPR V SC | DECS SC | S |  ;
DECS : DT V FORCMB DECS | DT V | V FORCMB ;
%%

yyerror()
{
        printf("\nError\n");
}

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

Exp 4 - PROCEDURE CALLS AND ARRAY REFERENCES IN C

LEX :-

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

%%
main                                return MAIN;
int|float|bool|"char*"              return DT;
return                              return RTN;
"&"                                 return REF;
"("                                 return OP;
")"                                 return CP;
"{"                                 return OC;
"}"                                 return CC;
";"                                 return SC;
","                                 return CM;
[0-9]+                              return NUM;
"+"|"-"|"*"|"/"|"++"|"--"|"="|"=="    return OPR;
(_|[a-zA-Z])(_|[a-zA-Z0-9])*        return ID;
\n                                  return 0;
%%
```

YACC :-

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

%token MAIN DT RTN REF OP CP OC CC SC CM NUM OPR ID
```

```
%start stmt

%%
stmt : S                              {printf("\nValid\n");};
S : FPROTO MAINF FDEF            ;
FPROTO : DT ID OP PMS CP SC FPROTO | ;
PMS : DT REFID CM PMS | DT REFID | ;
REFID : ID | REF ID        ;
MAINF : MAIN OP CP OC STMTS CC     ;
FDEF : DT ID OP PMS CP OC STMTS CC FDEF | ;
MPMS : REFID CM MPMS | NUM CM MPMS | REFID | NUM | ;
STMTS : STMTS STMTS | STMT | FDEC | ;
FDEC : ID OP MPMS CP SC | ;
STMT : STMT STMT | V O SC | V O V SC | V O V O V SC | ;
V : ID | NUM ;
O : OPR ;
%%

yyerror()
{
printf("\nError\n");
}

void main()
{
yyparse();
} */

%{
#include<stdio.h>
%}

%token MAIN DT RTN REF OP CP OC CC SC CM NUM OPR ID
%start stmt

%%
stmt : S                             {printf("\nValid\n");};
S : FPROTO MAINF FDEF          ;
FPROTO : DT ID OP PMS CP SC FPROTO | ;
PMS : DT REFID CM PMS | DT REFID | ;
REFID : REF ID | ID                   ;
MAINF : MAIN OP CP OC STMTS CC;
FDEF : DT ID OP PMS CP OC STMTS CC FDEF | ;
MPMS : REFID CM MPMS | NUM CM MPMS | REFID | NUM | ;
STMTS : STMT STMTS | FDEC STMTS | ;
FDEC : ID OP MPMS CP SC      ;
STMT : V O SC | V O V SC | V O V O V SC | ;
V : ID | NUM ;
O : OPR ;
```

```
%%

yyerror()
{
        printf("\nError\n");
}

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

Exp 5 - Calculating FIRST and FOLLOW of a Grammar

Calculating FIRST alone :-

C++

```
#include<iostream>
#include<bits/stdc++.h>
#include<string.h>

using namespace std;

set<char> compute_first(char symbol,map<char, vector<string>> mp,set<char> &vis)

{
    if(vis.find(symbol)!=vis.end())
    {
        set<char> dummy;
        return dummy;
    }

    vis.insert(symbol);

    set<char> first_set;

    vector<string> productions=mp[symbol];

    for(auto it:productions)
    {
        if(it.length()==0)
        {
            first_set.insert(' ');
        }
        else{
            char f_s=it[0];
            if(isupper(f_s))
            {
                set<char> med;
                med=compute_first(f_s,mp,vis);
                first_set.insert(med.begin(),med.end());
            }
            else
            {
                first_set.insert(f_s);
            }
        }
    }

    return first_set;
}

set<char> get_first(char symbol,map<char, vector<string> > mp)

{
    set<char> vis;

    return compute_first(symbol,mp,vis);

}


int main()
{
    map<char, vector<string> > mp;

    mp['S']={"B","bBA","C"};

    mp['A']={"B","e"};
    mp['B']={"d","e" ,"C"};
    mp['C']={"B","fA","g"};

    string sap="SABC";

    for(int i=0;i<sap.length();i++)
    {
        set<char> st=get_first(sap[i],mp);

        cout<<"first("<<sap[i]<<"):"<<endl;
        for(auto it: st)
        {
            cout<<it<<" ";
        }
cout<<endl;

}
```

```
     return 0;
    }
```

Python

```
# Function to find FIRST set for a variable
def find_first_set(variable, productions, first_sets, terminals_set, epsilon_set, visited):
    # If FIRST set for this variable is already computed, return it
    if variable in first_sets:
        return first_sets[variable]

    first_set = set()

    # Avoid infinite recursion on cyclic epsilon productions
    if variable in visited:
        return first_set

    visited.add(variable)

    # Iterate over productions
    for lhs, rhs in productions:
        if lhs == variable:
            for symbol in rhs:
                # If symbol is a terminal, add it to the FIRST set
                if symbol in terminals_set:
                    first_set.add(symbol)
                    break
                # If symbol is epsilon, add it to the FIRST set and continue to the next symbol
                elif symbol == 'e':
                    first_set.add('e')
                else:
                    # Recursively find FIRST set for the non-terminal symbol
                    non_terminal_first = find_first_set(symbol, productions, first_sets, terminals_set, epsilon_set,
visited)
                    # Add the computed FIRST set to the FIRST set of the current variable
                    first_set |= non_terminal_first
                    # If epsilon is not in the FIRST set of the non-terminal symbol, stop iterating
                    if 'e' not in non_terminal_first:
                        break

    # Cache the computed FIRST set
    first_sets[variable] = first_set
    visited.remove(variable)
    return first_set

# Main code
numprod = int(input("Enter number of productions: "))
productions = []
```

```
# Input productions
for i in range(numprod):
    production = input(f"Enter production {i + 1}: ").split("->")
    if len(production) != 2:
        print("Invalid production format. Please use the format 'A -> XYZ'.")
        continue
    lhs, rhs = production
    productions.append((lhs.strip(), rhs.strip()))

# Initialize terminals and variables
terminals = []
variables = []

# Extract terminals and variables from productions
for lhs, rhs in productions:
    variables.append(lhs)
    for symbol in rhs:
        if symbol.islower() and symbol not in terminals:
            terminals.append(symbol)

# Add additional terminals: arithmetic operators, parentheses, brackets, and square brackets
additional_terminals = ['+', '-', '*', '/', '(', ')', '[', ']', '{', '}']
terminals += additional_terminals

# Set of terminals
terminals_set = set(terminals)

# Set of variables with epsilon productions
epsilon_set = set(lhs for lhs, rhs in productions if rhs == 'e')

# Dictionary to store FIRST sets
first_sets = {}

# Compute FIRST sets for each variable
for variable in set(variables):  # Convert variables list to set to remove duplicates
    find_first_set(variable, productions, first_sets, terminals_set, epsilon_set, set())

# Print FIRST sets
for variable in set(variables):  # Convert variables list to set to remove duplicates
    print(f"FIRST({variable}) = {first_sets[variable]}")
```

{ INPUT FORMAT = VAR->(VAR U TER) [no space between variable and '->']* }

COUNTING NUMBER OF TOKENS

```
/*Lex code to count total number of tokens */
```

```
%{
int n = 0 ;
%}

// rule section
%%

//count number of keywords
"while"|"if"|"else" {n++;printf("\t keywords : %s", yytext);}

// count number of keywords
"int"|"float" {n++;printf("\t keywords : %s", yytext);}

// count number of identifiers
[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("\t identifier : %s", yytext);}

// count number of operators
"<="|"=="|"="|"++"|"-"|"*"|"+" {n++;printf("\t operator : %s", yytext);}

// count number of separators
[(){}|, ;] {n++;printf("\t separator : %s", yytext);}

// count number of floats
[0-9]*"."[0-9]+ {n++;printf("\t float : %s", yytext);}

// count number of integers
[0-9]+ {n++;printf("\t integer : %s", yytext);}

. ;
%%


int main()

{

        yylex();

        printf("\n total no. of token = %d\n", n);

}
```

COUNTING NUMBER OF WORDS,SPACES,CHARACTERS:-

```
%{
#include<stdio.h>
int lc=0,sc=0,tc=0,ch=0,wc=0;
%}
```

```
%%
[\n]         {lc++; ch+=yyleng;}
[" "\t]      {sc++; ch+=yyleng;}
[^\t]        {tc++; ch+=yyleng;}
[^\t\n]+    {wc++; ch+=yyleng;}
%%

void main()
{
    printf("Enter sentence : ");
    yylex();
    printf("\nNumber of lines : %d",lc);
    printf("\nNumber of spaces : %d",sc);
    printf("\nNumber of tabs : %d",tc);
    printf("\nNumber of words : %d",wc);
    printf("\nNumber of characters : %d",ch);
}
```

{After typing input, to get result, press Enter then Ctrl+D}

COUNTING NUMBER OF SINGLE LINE AND MULTI LINE COMMENTS

```
%{
#include<stdio.h>
int count,cont;
%}

%%
"//"    count++;
"/*"([^*]|[*][^/])*"*/"            {cont++;}
.    ;
\n    ;
%%

void main()
{
printf("\nEnter input = ");
yylex();
printf("\nNo. of single line comments = %d",count);
printf("\nNo of multi line comments = %d",cont);
}
```

{After typing input, to get result, press Enter then Ctrl+D}