1a. Lex file.l

```
%{
#include<stdio.h>
int lines=0,chars=0,words=0,spaces=0;
%}
%%
\n        {lines++;}
\t        {spaces+=4;}
[ ]       {spaces++;}
[^ \t\n#]+ {words++;chars+=yyleng;} //[^ \t\n#] not space, tab, line, #
# {return 0;}
%%
int yywrap()
{
        return 1;
}
int main()
{
        printf("Enter the string:\n");
        yylex();
        printf("Lines: %d\nWords: %d\nChars: %d\nSpaces: %d\n",lines,words,chars,spaces);
        return 0;
}
```

```
flex program1a.l
gcc lex.yy.c -o program1a -lfl
./program1a
```

Input: any string with numbers and special characters & (% @

(1b) lex file

```
%{
#include "y.tab.h"
%}
%%
"a"	{return 'a';}
"b"	{return 'b';}
"c" {return 'c';}
.	{return yytext[0];}
\n	{return 0;}
%%
```

(yacc file).y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yyerror();
int yylex();
%}
%%
S:A B
;
A:'a'A'b'
|
;
B:'b'B'c'
|
;
%%
int main()
{
	printf("Enter the input:\n");
	yyparse();
	printf("Valid string\n");
}
int yyerror()
{
	printf("Invalid string\n");
	exit(0);
}
```

```
yacc -d pg1b.y
flex pg1b.l
gcc y.tab.c lex.yy.c -o output -lfl
./output
```

Input:

```
  aabbcc
  abc
  ab
  bc
```

(2a): lex file

```
%{
#include <stdio.h>
int posint = 0, negint = 0, posfrac = 0, negfrac = 0;
%}

num [0-9]+

posint \+?{num}
negint -{num}

posnum \+?{num}\/\+?{num}|-{num}\/-{num}
negnum -{num}\/\+?{num}|\+?{num}\/-{num}

%%

{posint} posint++;

{negint} negint++;

{posnum} posfrac++;

{negnum} negfrac++;

[ \t] ;

[\n] return 0;
```

```
. ECHO;

%%

int yywrap(){}

int main() {
	yylex();
	printf("Positive integers: %d\n", posint);
	printf("Negative integers: %d\n", negint);
	printf("Positive fractions: %d\n", posfrac);
	printf("Negative fractions: %d\n", negfrac);
}
```

Input:: 1 -2 3/4 -5/6 7/-8 -9/-10 +11/12

(2b): 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>
	int yylex();
	int yyerror();

%}
%token NUM
%left '+' '-'
%left '/' '*'
%%
S:I {printf("Result is %d\n",$$);}
```

```
;
I:I'+'I    {$$=$1+$3;}
|I'-'I              {$$=$1-$3;}
|I'*'I              {$$=$1*$3;}
|I'/'I              {if($3==0){yyerror();}   else{$$=$1/$3;}}
|'('I')'    {$$=$2;}
|NUM                {$$=$1;}
|'-'NUM             {$$=-$2;}
;
%%
int main()
{
        printf("Enter operation:\n");
        yyparse();
        printf("Valid\n");
        return 0;
}
int yyerror()
{
        printf("Invalid\n");
        exit(0);
}
```

i/p: 1+2*3/4-5

(3a) lex file

```
%{

#include<stdio.h>

int flag=0;

int c=0;

int flg=0;

%}

%%

"//".* {
```

```
if(flag==1) {
fprintf(yyout," ");
flg--;
}
else{
c++;
fprintf(yyout," ");
flg++;
}
}

"/*".*\n?"*/"? {
if(flg==1){
fprintf(yyout," ");
}
else {
flag++;
fprintf(yyout," ");
c++;}}

.*"*/" {if(flag==1){ fprintf(yyout," "); c++;flag--;}}

%%

int main()

{

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

yyout = fopen("v1.txt","w");

yylex();

printf("Number of comment lines=%d",c);

}
int yywrap()

{
return 1;
}
```

Input:
//cg
//comment

(3b) lex file

```
%{
	#include "y.tab.h"
%}
%%
"for" return FOR;
"int"|"float"|"double"|"bool" return TYPE;
">"|"<"|">="|"<="|"=="|"!=" return OP;
[a-zA-Z]* return IDEN;
[0-9]+ return NUM;
[\n\t ]  ;
.   return yytext[0];
%%
```

(yacc file)

```
%{
	#include<stdio.h>
	#include<stdlib.h>
	int yylex();
	int yyerror();
	int cnt=0;
%}
%token FOR IDEN NUM TYPE OP
%left '+' '-'
%left '*' '/'
%%

// Tokens

// FOR -> for
// IDEN -> identifier
// NUM -> number
// TYPE -> datatype
// OP -> relational operator

// Non-terminals

// S -> Start symbol
// BODY -> Body  of For loop
// COND -> Condition
// S1 -> Single Statement
// SS -> Set of statements
// T -> Term
// E -> Expression
// F -> For loop block
// DA -> Declaration or assignment
// DECL -> Declaration
// ASSGN -> Assignment

S:F;
F:FOR'('DA';'COND';'S1')'BODY { cnt++; } |
  FOR'(' ';'COND';'S1')'BODY { cnt++; } |
  FOR'('DA';' ';'S1')'BODY { cnt++; } |
  FOR'(' ';' ';'S1')'BODY { cnt++; } ;

DA:DECL|ASSGN
DECL: TYPE IDEN | TYPE ASSGN;
ASSGN : IDEN '=' E;
COND : T OP T;
T : NUM | IDEN ;

BODY: S1';' | '{'SS'}' | F |';';

SS: S1 ';' SS | F SS |;
S1: ASSGN | E | DECL ;
E : E '+' E | E '-' E | E '*' E | E '/' E | '-''-'E | '+''+'E | E'+''+' | E'-''-' | T ;
%%
int main()
{
        printf("Enter the snippet:\n");
        yyparse();
        printf("Count of for : %d\n",cnt);
        return 0;
}
int yyerror()
{
```

```
            printf("Invalid\n");
            exit(0);
}
```

Input:

```
for(int i = 0; i < 5; i++) {
    for(int j = 0; j < 5; j++) {
        x = x + 1;
    }
}
```

(4a) lex file

```
%{

#include<stdio.h>

int i=0,k=0,op=0;

%}

%%

auto|break|case|char|continue|do|default|const|double|else|enum|extern|for|if|goto|float|int|long|r
egister|return|signed|static|sizeof|short|struct|switch|typedef|union|void|while|volatile|unsigned { }

("/"[^\"]*"/") { k++;}

(""|[a-z]|[A-Z])(""|[a-z]|[A-Z]|[0-9])* {i++;}

"#include".* ;

"#"[a-zA-Z]+.* ;

[;] ;

[ ] ;
```

```
[,] ;

[+*%/-] {op++;}

[\n] ;

%%

void main()

{

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

yylex();

printf("No. of identifiers=%d\n,keywords=%d,operators=%d",i,k,op);

}

int yywrap()

{ return 1;

}
```

d.c

```
#include <stdio.h>
int main() {
    int i = 0;
    while (i < 10) {
        i++;
    }
    return 0;
}
```

(5) lex file

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

```
%%
"int" { return INT; }
"float" { return FLOAT; }
"char" { return CHAR; }
"double" { return DOUBLE; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return IDENTIFIER; }
[0-9]+ {return NUM;}
"[" { return '['; }
"]" { return ']'; }
"," { return ','; }
";" { return ';'; }
[ \t\n] { /* Ignore whitespace */ }
. { /* Ignore any other characters */ }

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

(yacc file)

```
%{
#include <stdio.h>
#include <stdlib.h>
int var_count = 0;
void yyerror(const char *s);
int yylex();
%}

%union {
char *str;
}

%token <str> IDENTIFIER
%token INT FLOAT CHAR DOUBLE NUM

%%
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;
}
```

Input: int a,b,c;

8 lex file

```
%{
#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;
}
```

8.yacc file

```
%{
#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 SEMI { // Handling a = b;
        printf("movl %s, %%eax\n", $3);
        printf("movl %%eax, %s\n", $1);
    }
    | 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;
}
```

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