1a. Leex file

```
%{
#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)

```
%{
#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:: 1 -2 3/4 -5/6 7/-8 -9/-10 +11/12

(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);
}
```

(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;}}
```

```
|'('l')'    {$$=$2;}
|NUM            {$$=$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 "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(a;b;c){}
for(a;b;c){for(a;b;c){d;}}

(3b) lex file

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

%%
[\t\n ] ;

"int"|"float"|"char"|"void" {return TYPE;}

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

[0-9]+ {return NUM;}

. {return yytext[0];}
%%
```

(yacc file)

```
%{
```

```
   #include<stdio.h>
   #include<stdlib.h>
   int yyerror();
   int yylex();
%}

%token TYPE IDEN NUM
%left '+' '-'
%left '*' '/'

%%
// Tokens

// IDEN -> identifier
// NUM -> number
// TYPE -> datatype

// Non-terminals

// S -> Start symbol
// FUN -> function block
// PARAMS -> parameters
// PARAM -> parameter
// BODY -> Function body
// S1 -> Single Statement
// SS -> Set of statements
// T -> Term
// E -> Expression
// DECL -> Declaration
// ASSGN -> Assignment

S: FUN  { printf("Accepted\n"); exit(0); } ;
FUN: TYPE IDEN '(' PARAMS ')' BODY ;
BODY: S1';' | '{'SS'}'
PARAMS: PARAM','PARAMS | PARAM | ;
PARAM:  TYPE IDEN;
SS: S1';'SS | ;
S1: ASSGN | E | DECL ;
DECL: TYPE IDEN | TYPE ASSGN ;
ASSGN : IDEN '=' E ;
E : E '+' E | E '-' E | E '*' E | E '/' E | '-''-'E | '+''+'E | E'+''+' | E'-''-' | T ;
T : NUM | IDEN ;
%%
int main()
```

```
{
   printf("enter input: ");
   yyparse();
   printf("successfull\n");
   return 0;
}
int yyerror()
{
   printf("ERROR\n");
   exit(0);
}
```

Input: a=b+c*d
a+b+c+d+e+f
a=(b+d)*(c+e)

(4) lex file

```
%{
#include <string.h>

#include "y.tab.h"
%}


%%

[a-zA-Z_][a-zA-Z_0-9]* {
        yylval.exp = strdup(yytext);
        return IDEN;
}

[0-9]+ {
        yylval.exp = strdup(yytext);
        return NUM;
}

[-+*/] return yytext[0];
```

```
[()=] return yytext[0];

[\n]+ return '\n';

[ \t]+ ;

. ;

%%
```

(yacc file)

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror();

extern FILE *yyin; // optional


typedef char *string;

struct {
        string res, op1, op2;
        char op;
} code[100];
int idx = -1;

string addToTable(string, string, char);
void threeAddressCode();
void quadruples();
%}

%union {
        char *exp;
}

%token <exp> IDEN NUM
%type <exp> EXP

%left '+' '-'
```

```
%left '*' '/'

%%

STMTS           : STMTS STMT
                |
                ;

STMT  : EXP '\n'
                ;

EXP     : EXP '+' EXP { $$ = addToTable($1, $3, '+'); }
                | EXP '-' EXP { $$ = addToTable($1, $3, '-'); }
                | EXP '*' EXP { $$ = addToTable($1, $3, '*'); }
                | EXP '/' EXP { $$ = addToTable($1, $3, '/'); }
                | '(' EXP ')' { $$ = $2; }
                | IDEN '=' EXP { $$ = addToTable($1, $3, '='); }
                | IDEN { $$ = $1; }
                | NUM { $$ = $1; }
                ;

%%

int yyerror() {
        printf("Error");
        exit(0);
}

int main() {
        // yyin = fopen("6.txt", "r");
        // Only if input is given from text file
        yyparse();

        printf("\nThree address code:\n");
        threeAddressCode();

        printf("\nQuadruples:\n");
        quadruples();
}


string addToTable(string op1, string op2, char op) {
        if(op == '=') {
                code[idx].res = op1;
```

```
                return op1;
        }

        idx++;
        string res = malloc(3);
        sprintf(res, "@%c", idx + 'A');
        code[idx].op1 = op1;
        code[idx].op2 = op2;
        code[idx].op = op;
        code[idx].res = res;
        return res;
}

void threeAddressCode() {
        for(int i = 0; i <= idx; i++) {
                printf("%s = %s %c %s\n", code[i].res, code[i].op1, code[i].op, code[i].op2);
        }
}


void quadruples() {
        for(int i = 0; i <= idx; i++) {
                printf("%d:\t%s\t%s\t%s\t%c\n", i, code[i].res, code[i].op1, code[i].op2, code[i].op);
        }
}
```

input : a=b+c*d
a+b+c+d+e+f
a=(b+d)*(c+e)

(5) lex file

```
%{
#include <string.h>

#include "y.tab.h"
%}


%%

[a-zA-Z_][a-zA-Z_0-9]* {
```

```
	yylval.exp = strdup(yytext);
	return IDEN;
}

[0-9]+ {
	yylval.exp = strdup(yytext);
	return NUM;
}

[-+*/] return yytext[0];

[()=] return yytext[0];

[\n]+ return '\n';

[ \t]+ ;

. ;

%%
```

(yacc file)

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror();

//extern FILE *yyin;

typedef char *string;

struct {
	string res, op1, op2;
	char op;
} code[100];
int idx = -1;

string addToTable(string, string, char);
void targetCode();
%}
```

```
%union {
	char *exp;
}

%token <exp> IDEN NUM
%type <exp> EXP

%left '+' '-'
%left '*' '/'

%%

STMTS		: STMTS STMT
		|
		;

STMT	: EXP '\n'
		;

EXP	: EXP '+' EXP { $$ = addToTable($1, $3, '+'); }
		| EXP '-' EXP { $$ = addToTable($1, $3, '-'); }
		| EXP '*' EXP { $$ = addToTable($1, $3, '*'); }
		| EXP '/' EXP { $$ = addToTable($1, $3, '/'); }
		| '(' EXP ')' { $$ = $2; }
		| IDEN '=' EXP { $$ = addToTable($1, $3, '='); }
		| IDEN { $$ = $1; }
		| NUM { $$ = $1; }
		;

%%

int yyerror(const char *s) {
	printf("Error %s", s);
	exit(0);
}

int main() {
	//yyin = fopen("8.txt", "r");
	yyparse();

	printf("\nTarget code:\n");
	targetCode();
}
```

```
string addToTable(string op1, string op2, char op) {
	if(op == '=') {
		code[idx].res = op1;
		return op1;
	}

	idx++;
	string res = malloc(3);
	sprintf(res, "@%c", idx + 'A');

	code[idx].op1 = op1;
	code[idx].op2 = op2;
	code[idx].op = op;
	code[idx].res = res;
	return res;
}

void targetCode() {
	for(int i = 0; i <= idx; i++) {
		string instr;
		switch(code[i].op) {
		case '+': instr = "ADD"; break;
		case '-': instr = "SUB"; break;
		case '*': instr = "MUL"; break;
		case '/': instr = "DIV"; break;
		}

		printf("LOAD\t R1, %s\n", code[i].op1);
		printf("LOAD\t R2, %s\n", code[i].op2);
		printf("%s\t R3, R1, R2\n", instr);
		printf("STORE\t %s, R3\n", code[i].res);
	}
}
```

Input.txt:
c = d + e
a=b+c
f=a-b

Input : ./outputfilename < input.txt

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