Sr. No.	File Names	Topic (Short Description)
1.	assign1.l pro.c	Lexical analyzer for subset of C (token generator)
2.	assign2.l sample.txt	Count words, characters, and lines from input text
3.	assign3.l, assign3.y	YACC - Validate variable declarations in C
4.	assign4.l, assign4.y	YACC - Calculator for arithmetic expressions
5.	assign5.c	Recursive Descent Parser (RDP)
6.	assign6.l, assign6.y	YACC - Recognize simple & compound sentences
7.	assign7.l, assign7.y	Lex & YACC - Symbol table generation
8.	assign8.l, assign8.y	Intermediate code gen: 3-address & quadruple formats

..Assignment No.1..

Q1: Write a program using LEX specifications to implement lexical analysis phase of compiler to generate tokens of subset of C program.

assign1.l

```
%{
#include <stdio.h>
FILE *yyin;
%}

%%
"int"|"float"|"void"|"return"|"main"  { printf("Keyword: %s\n", yytext); }
"="|"+"|"-"|"*"|"/"                   { printf("Operator: %s\n", yytext); }
"(" | ")" | "{" | "}" | ";" | ","     { printf("Symbol: %s\n", yytext); }
[0-9]+\.[0-9]+                        { printf("Decimal Number: %s\n", yytext); }
[0-9]+                                { printf("Integer: %s\n", yytext); }
\"[^\"]*\"                            { printf("String Literal: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*                { printf("Identifier: %s\n", yytext); }
[ \t\n]                               { /* Ignore whitespace */ }
.                                     { printf("Unknown: %s\n", yytext); }
%%

int yywrap() {
```

```
    return 1;
}

int main() {
    yyin = fopen("pro.c", "r");
    yylex();
    return 0;
}
```

pro.c

```
int main() {
    int a = 10;
    float b = 20.5;
    printf("Hello World");
    return 0;
}
```

How to Run

```
lex assign1.l
gcc lex.yy.c
./a.out
```

..Assignment 2..

Write a LEX program to display word, character, and line counts for a sample input text file.

assign2.l

```
%{
int word_count = 0, char_count = 0, line_count = 0;
%}

%%
[ \t]+            { /* Skip spaces and tabs */ }
\n                { line_count++; char_count++; }
[a-zA-Z0-9_]+     { word_count++; char_count += yyleng; }
.                 { char_count++; }
%%

int yywrap() {
   return 1;
}

int main() {
   yylex();
   printf("Words: %d\n", word_count);
   printf("Characters: %d\n", char_count);
   printf("Lines: %d\n", line_count);
   return 0;
```

}

sample.txt

Hello world!

This is a test file.

Lex is fun.

🧪 Steps to Compile and Run on Ubuntu

lex assign2.l # Step 1: Generate C code (lex.yy.c)

gcc lex.yy.c # Step 2: Compile

./a.out < sample.txt # Step 3: Provide file input

..Assignment No 3..

Write a program using YACC specifications to implement the syntax analysis phase of a compiler to validate the type and syntax of variable declarations in a C program.

assign3.l

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

%%

int             { return INT; }
float           { return FLOAT; }
char            { return CHAR; }
[a-zA-Z_][a-zA-Z0-9_]*    { yylval = strdup(yytext); return ID; }
;               { return SEMI; }
[ \t\n]         { /* Ignore whitespace */ }

.               { return yytext[0]; }

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

assign3.y

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

%token INT FLOAT CHAR ID SEMI

%%
stmt_list: stmt_list stmt
        | stmt
        ;

stmt: type ID SEMI    { printf("Valid declaration: %s\n", $2); }
    ;

type: INT
    | FLOAT
    | CHAR
    ;
%%
```

```
void yyerror(const char *s) {
    printf("Syntax Error: %s\n", s);
}

int main() {
    printf("Enter variable declarations (e.g., int x; float y;):\n");
    yyparse();
    return 0;
}
```

Steps to Compile and Run on Ubuntu

```
yacc -d assign3.y       # Step 1: Generate y.tab.c and y.tab.h
lex assign3.l           # Step 2: Generate lex.yy.c
gcc lex.yy.c y.tab.c -o assign3  # Step 3: Compile everything
./assign3               # Step 4: Run the program
```

Sample Input (during execution)

```
int a;
float b;
char name;
```

..Assignment No. 4..

Write a program using YACC specifications to implement a calculator to perform various arithmetic operations.

assign4.l

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

%%

[0-9]+      { yylval = atoi(yytext); return NUM; }
[\n]        { return EOL; }
[+\-*/()]   { return yytext[0]; }
[ \t]       { /* Ignore spaces */ }

.           { return yytext[0]; }

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

assign4.y

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

%token NUM
%token EOL

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

%%
calclist:
   | calclist expr EOL { printf("Result = %d\n", $2); }
   ;

expr:
     NUM              { $$ = $1; }
   | expr '+' expr     { $$ = $1 + $3; }
   | expr '-' expr     { $$ = $1 - $3; }
   | expr '*' expr     { $$ = $1 * $3; }
   | expr '/' expr     {
                        if ($3 == 0) {
```

```
                printf("Division by zero error\n");
                exit(1);
            } else {
                $$ = $1 / $3;
            }
        }
    | '(' expr ')'     { $$ = $2; }
    ;
%%

int yyerror(char *s) {
    printf("Syntax Error: %s\n", s);
    return 0;
}

int main() {
    printf("Enter arithmetic expressions (press Ctrl+D to exit):\n");
    yyparse();
    return 0;
}
```

Steps to Compile and Run on Ubuntu

```
yacc -d assign4.y          # Step 1: Generate parser code
```

lex assign4.l # Step 2: Generate lexer code

gcc lex.yy.c y.tab.c -o assign4 # Step 3: Compile

./assign4 # Step 4: Run

Sample Input (during execution)

2 + 3 * 4

(1 + 2) * 5

..Assignment No.5..

Write a program to implement a Recursive Descent Parser (RDP) for a simple grammar.

We'll use a basic grammar for arithmetic expressions like:

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

This grammar can parse expressions like: id + id * id

- **E is the starting symbol**
- **ε (epsilon) represents an empty string**
- **id is a valid identifier (e.g., a, x, etc.)**

assign5.c

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>


char input[100];

int i = 0;


// Function declarations

void E();

void Eprime();

void T();
```

```
void Tprime();
void F();

void error() {
    printf(" ✕ Error in parsing at position %d\n", i);
    exit(1);
}

void match(char expected) {
    if (input[i] == expected) {
        i++;
    } else {
        error();
    }
}

void E() {
    T();
    Eprime();
}

void Eprime() {
    if (input[i] == '+') {
        match('+');
```

```
        T();
        Eprime();
    }
    // else ε (do nothing)
}

void T() {
    F();
    Tprime();
}

void Tprime() {
    if (input[i] == '*') {
        match('*');
        F();
        Tprime();
    }
    // else ε (do nothing)
}

void F() {
    if (input[i] == '(') {
        match('(');
        E();
```

```
        match(')');
    } else if (isalpha(input[i])) {
        match(input[i]);  // assuming single char id
    } else {
        error();
    }
}

int main() {
    printf("Enter an expression (e.g., a+b*c): ");
    scanf("%s", input);

    E();

    if (input[i] == '\0') {
        printf("✅ Input is successfully parsed!\n");
    } else {
        printf("❌ Unexpected characters at the end\n");
    }

    return 0;
}
```

Steps to Compile and Run on Ubuntu

gcc assign5.c -o assign5

Sample Input & Output

a+b*c

./assign5

..Assignment No 6..

Write a program using YACC specifications to implement the syntax analysis phase of a compiler to recognize simple and compound sentences given in an input file. Display the type of sentence.

assign6.l

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

%%
[aA] | [tT]he          { return ARTICLE; }
boy | girl | cat | dog  { return NOUN; }
sees | likes | eats     { return VERB; }
and | but | or         { return CONJ; }
\n                     { return EOL; }
[ \t]                  { /* Ignore whitespace */ }
.                      { /* Ignore other chars */ }
%%
int yywrap() {
   return 1;
}
```

assign6.y

```
%{
#include <stdio.h>
int simple = 0, compound = 0;
%}

%token ARTICLE NOUN VERB CONJ EOL

%%
start:
   sentence EOL {
      if (compound)
         printf("✅ Compound Sentence\n");
      else if (simple)
         printf("✅ Simple Sentence\n");
      else
         printf("❌ Not a valid sentence\n");
      simple = compound = 0;
   }
```

```
    ;

sentence:
      NP VP          { simple = 1; }
    | sentence CONJ sentence { compound = 1; }
    ;

NP:
    ARTICLE NOUN
    ;

VP:
    VERB NP
    ;
%%

int yyerror(char *s) {
    printf(" ✕ Syntax Error: %s\n", s);
    return 0;
}
```

```
int main() {
    printf("Enter a sentence (end with Enter):\n");
    yyparse();
    return 0;
}
```

Compile and Run on Ubuntu

```
yacc -d assign6.y
lex assign6.l
gcc lex.yy.c y.tab.c -o assign6
./assign6
```

Sample Inputs

1. the boy sees a girl
    - Output: ✅ Simple Sentence
2. the boy sees a girl and the dog eats the cat
    - Output: ✅ Compound Sentence
3. boy the sees
    - Output: ❌ Not a valid sentence

..Assignment no 7..

Write a program using LEX and YACC to generate a symbol table for identifiers declared in a C-like program

assign7.l

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

%%
int          { return INT; }
float        { return FLOAT; }
char         { return CHAR; }
[a-zA-Z_][a-zA-Z0-9_]*   { yylval.str = strdup(yytext); return ID; }
[,]          { return COMMA; }
[;]          { return SEMICOLON; }
[ \t\n]      { /* Ignore whitespace */ }
.            { /* Ignore other characters */ }
%%

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

assign7.y

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

struct symbol {
    char name[100];
    char type[10];
} table[100];

int count = 0;
char current_type[10];

void insert(char *name) {
    for (int i = 0; i < count; i++) {
        if (strcmp(table[i].name, name) == 0) return; // already exists
    }
    strcpy(table[count].name, name);
    strcpy(table[count].type, current_type);
    count++;
}
%}
```

```
%union {
   char *str;
}

%token <str> ID
%token INT FLOAT CHAR
%token COMMA SEMICOLON

%%
program: declarations ;

declarations:
   declarations declaration
   | declaration
   ;

declaration:
     type id_list SEMICOLON
   ;

type:
     INT   { strcpy(current_type, "int"); }
   | FLOAT { strcpy(current_type, "float"); }
   | CHAR  { strcpy(current_type, "char"); }
```

```
    ;

id_list:
      ID                { insert($1); }
    | id_list COMMA ID   { insert($3); }
    ;
%%

int yyerror(char *s) {
    printf(" ❌ Error: %s\n", s);
    return 0;
}

int main() {
    printf("Enter variable declarations (e.g., int a, b; float x;):\n");
    yyparse();
    printf("\n🔍 Symbol Table:\n");
    printf("%-10s %-10s\n", "Name", "Type");
    for (int i = 0; i < count; i++) {
        printf("%-10s %-10s\n", table[i].name, table[i].type);
    }
    return 0;
}
```

Steps to Compile and Run on Ubuntu

yacc -d assign7.y

lex assign7.l

gcc lex.yy.c y.tab.c -o assign7

./assign7

Sample Input

int a, b;

float x;

char c;

..Assignment No 8..

Write a program using LEX and YACC to generate Intermediate Code in the form of Three Address Code (3AC) and Quadruple form for an assignment statement.

assign8.l

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

%%
[a-zA-Z][a-zA-Z0-9]* { yylval.str = strdup(yytext); return ID; }
[0-9]+               { yylval.str = strdup(yytext); return NUM; }
[=]                  { return ASSIGN; }
[+]                  { return PLUS; }
[-]                  { return MINUS; }
[*]                  { return MUL; }
[/]                  { return DIV; }
[;]                  { return SEMICOLON; }
[ \t\n]              { /* skip spaces */ }
.                    { /* skip unknown */ }
%%

int yywrap() {
   return 1;
```

```
}
```

assign8.y

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

int tempCount = 1;

typedef struct {
    char result[10];
    char arg1[10];
    char arg2[10];
    char op[3];
} Quad;

Quad quads[20];
int quadIndex = 0;

char* newTemp() {
    static char temp[10];
    sprintf(temp, "t%d", tempCount++);
    return strdup(temp);
}
```

```
%}

%union {
   char *str;
}

%token <str> ID NUM
%token ASSIGN PLUS MINUS MUL DIV SEMICOLON

%type <str> expr term factor

%%
stmt:
   ID ASSIGN expr SEMICOLON {
      printf("\nThree Address Code:\n");
      printf("%s = %s\n", $1, $3);

      printf("\nQuadruple Representation:\n");
      for (int i = 0; i < quadIndex; i++) {
         printf("%2d: (%s, %s, %s, %s)\n", i+1,
            quads[i].op, quads[i].arg1, quads[i].arg2, quads[i].result);
      }

      printf("%2d: (=, %s, -, %s)\n", quadIndex + 1, $3, $1);
```

```
    }
    ;

expr:
    expr PLUS term {
        char* temp = newTemp();
        printf("%s = %s + %s\n", temp, $1, $3);

        strcpy(quads[quadIndex].result, temp);
        strcpy(quads[quadIndex].arg1, $1);
        strcpy(quads[quadIndex].arg2, $3);
        strcpy(quads[quadIndex].op, "+");
        quadIndex++;

        $$ = temp;
    }
    | expr MINUS term {
        char* temp = newTemp();
        printf("%s = %s - %s\n", temp, $1, $3);

        strcpy(quads[quadIndex].result, temp);
        strcpy(quads[quadIndex].arg1, $1);
        strcpy(quads[quadIndex].arg2, $3);
        strcpy(quads[quadIndex].op, "-");
```

```
        quadIndex++;

        $$ = temp;
    }
    | term {
        $$ = $1;
    }
    ;

term:
    term MUL factor {
        char* temp = newTemp();
        printf("%s = %s * %s\n", temp, $1, $3);

        strcpy(quads[quadIndex].result, temp);
        strcpy(quads[quadIndex].arg1, $1);
        strcpy(quads[quadIndex].arg2, $3);
        strcpy(quads[quadIndex].op, "*");
        quadIndex++;

        $$ = temp;
    }
    | term DIV factor {
        char* temp = newTemp();
```

```
        printf("%s = %s / %s\n", temp, $1, $3);

        strcpy(quads[quadIndex].result, temp);
        strcpy(quads[quadIndex].arg1, $1);
        strcpy(quads[quadIndex].arg2, $3);
        strcpy(quads[quadIndex].op, "/");
        quadIndex++;

        $$ = temp;
    }
    | factor {
        $$ = $1;
    }
    ;

factor:
    ID {
        $$ = $1;
    }
    | NUM {
        $$ = $1;
    }
    ;
%%
```

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

int main() {
    printf("Enter assignment statement (e.g. a = b + c * d;):\n");
    yyparse();
    return 0;
}
```

Steps to Run on Ubuntu

```
yacc -d assign8.y
lex assign8.l
gcc lex.yy.c y.tab.c -o assign8
./assign8
```

Sample Input:

```
a = b + c * d;
```