# Formal Automata and Automata Theory Assignment – IT 321

**Name: Midanka Lahon**

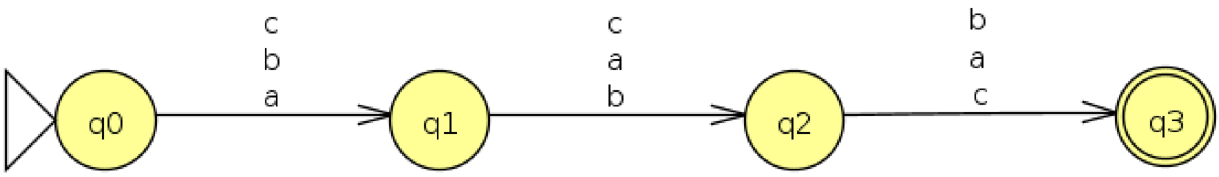
**Roll No: 220103009**

**Semester: 6th** **Branch: CSE**

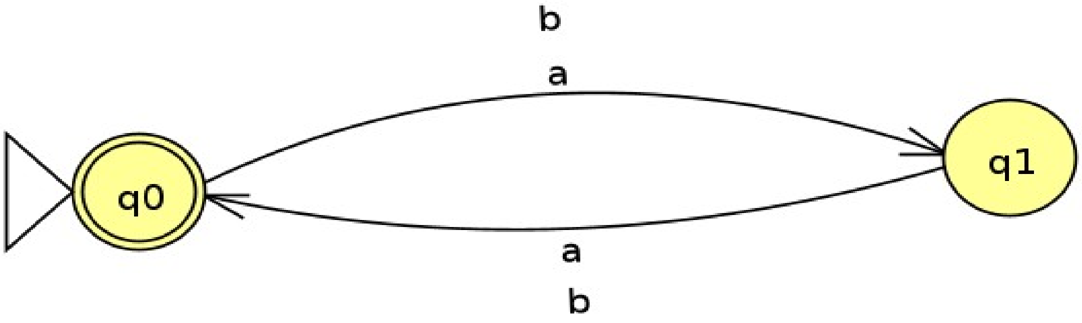
# SIMULATIONS:

1. { a, b, c } | |w| = 3
2. Z = { a, b } | |Z| mod 2 = 0
3. Z = { a, b, c } | |W| ⩽ 3
4. Z = { a, b, c } | |Z| mod 2 = 1
5. Σ = { 0, 1 } where 0 Even 1 Even
6. Σ = { 0, 1 } where 0 Odd 1 Odd
7. Starting with ‘a’ Σ = { a, b }
8. Starting with ‘b’ Σ = { a, b }
9. a*b Σ = { a, b }
10. a*b*

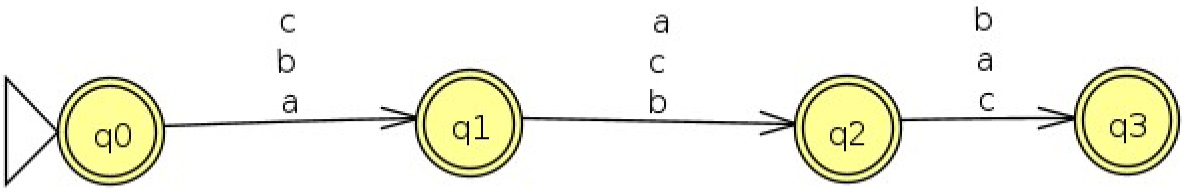
1. { a, b, c } | |w| = 3



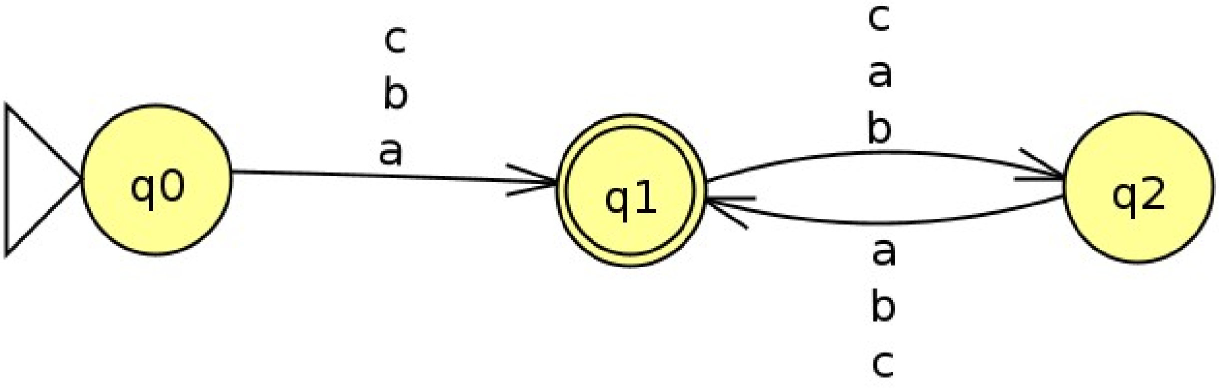
2. Z = { a, b } | |Z| mod 2 = 0



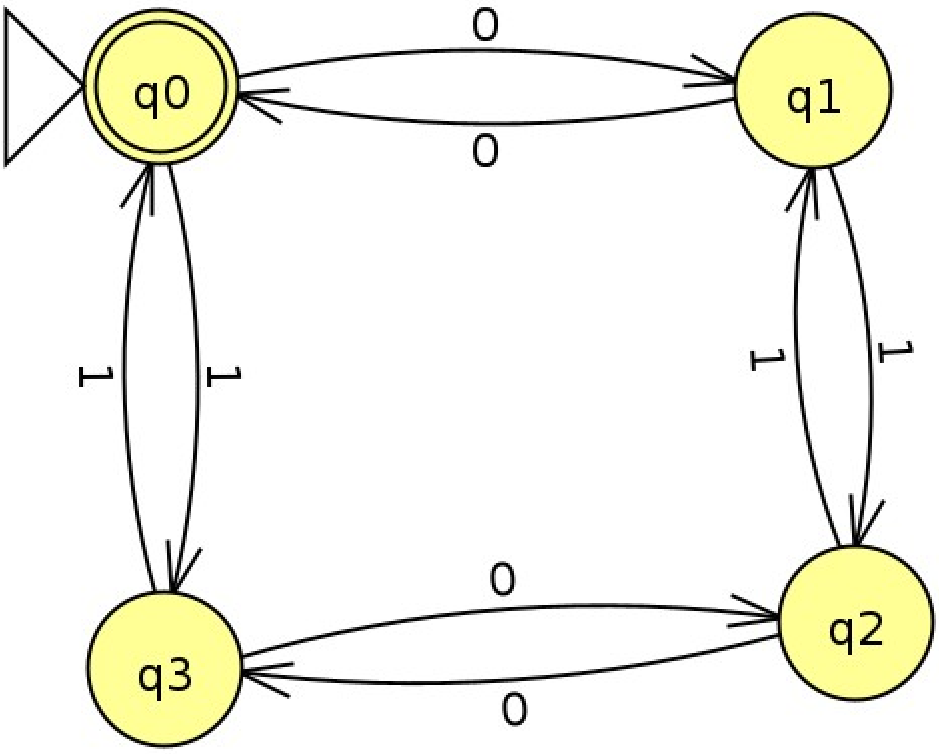
3. Z = { a, b, c } | |W| <= 3



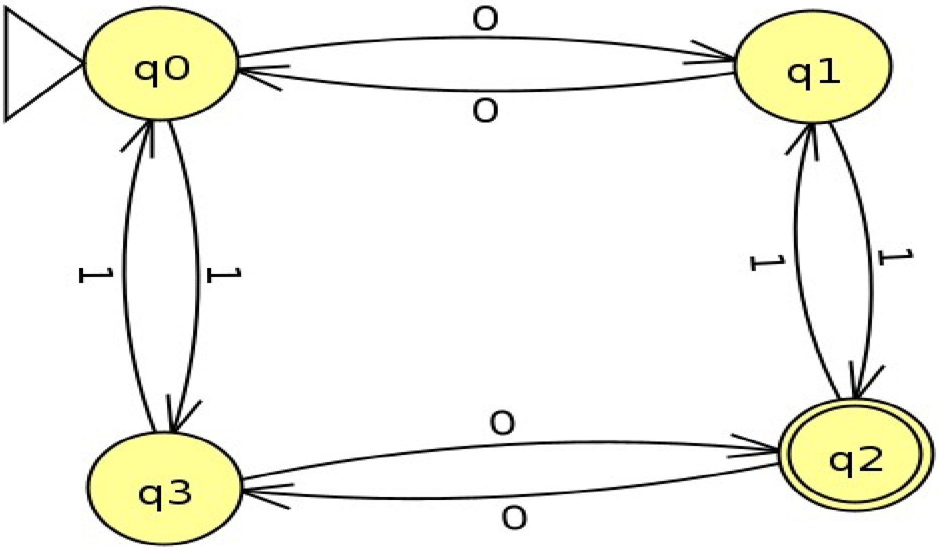
4. Z = { a, b, c } | |Z| mod 2 = 1



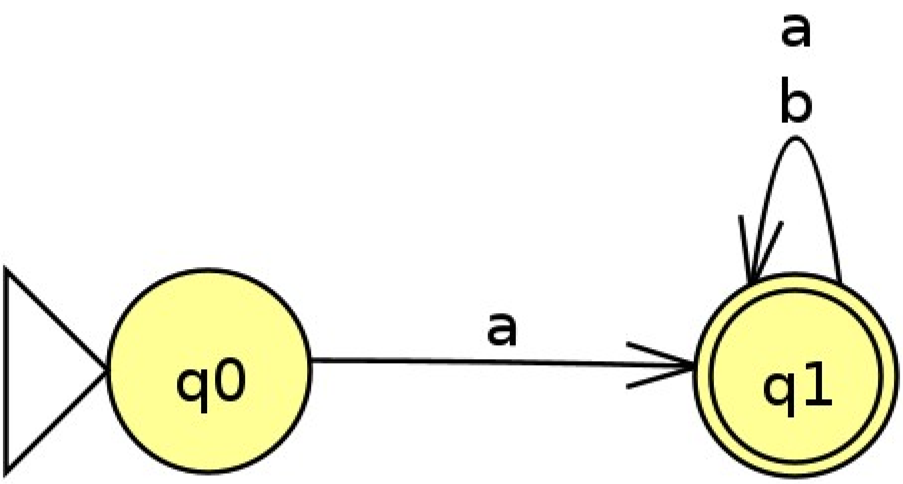
5. Σ = { 0, 1 } where 0 Even 1 Even



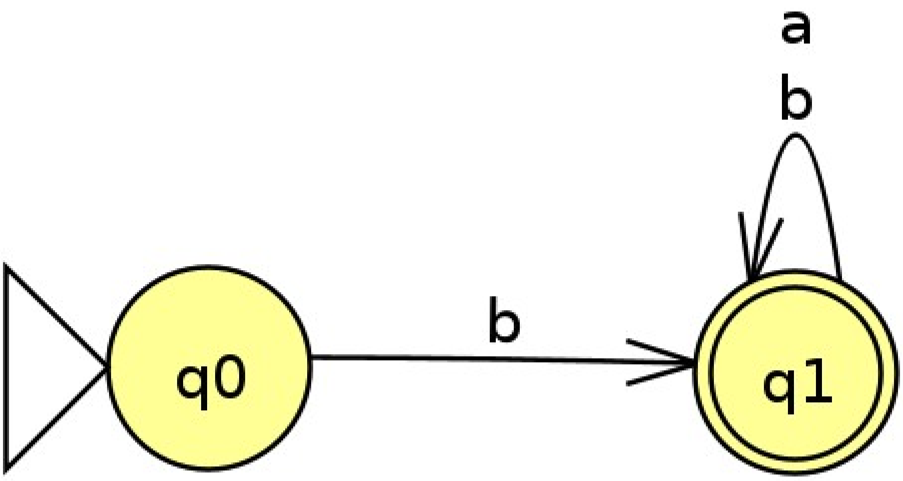
**6. Σ = { 0, 1 } where 0 Odd 1 Odd**



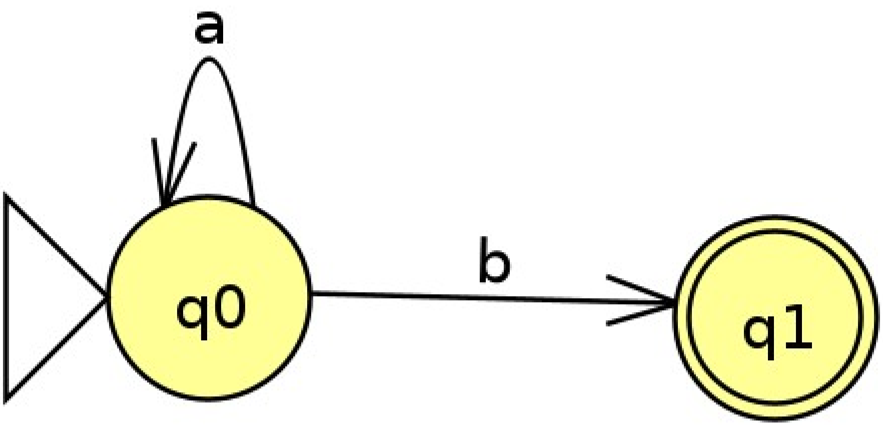
**7. Starting with ‘a’  Σ = { a, b }**



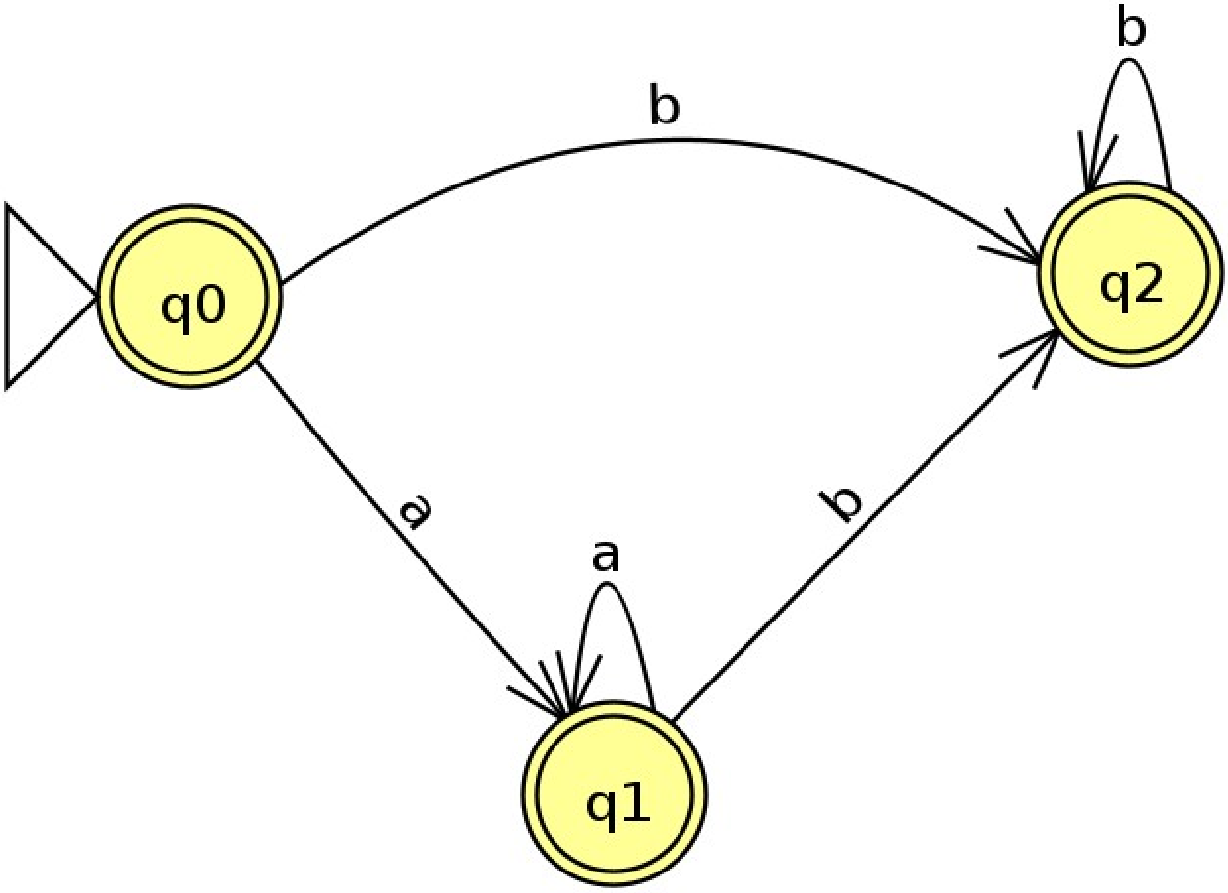
8. Starting with ‘b’  Σ = { a, b }



9. a*b  Σ = { a, b }



**10. a*b***



# LEX PROGRAMS

1. LEX Program to count the numbers of lines, spaces, text, characters, words.

2. LEX Program to identify capitalized words from a string.

3. LEX Program to identify capitalized lettes from a string.

4. LEX Program to count number of commented lines.

5. LEX Program to recognize all valid arithmetic sequence.

6. LEX Program to count the number of words.

7. LEX Program to count the number of tokens.

8. LEX Program to count no. Of lines in text file.

9. LEX Program to detect vowel and consonants.

10. LEX Program to count no. of characters in text file.

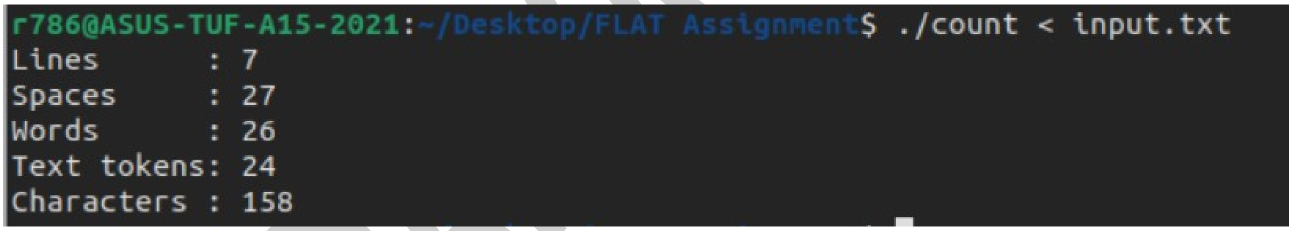
## 1. LEX Program to count the numbers of lines, spaces, text, characters, words.

### Program:

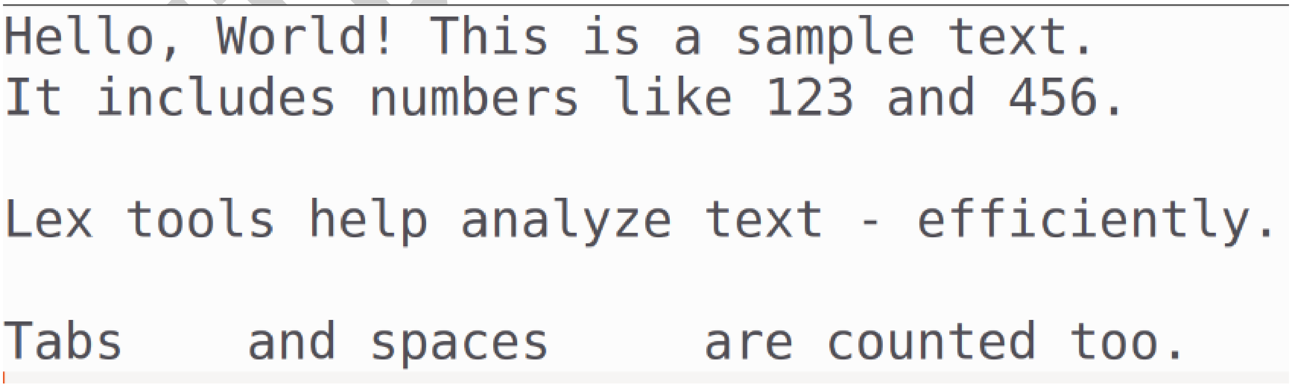
```
%{
#include <stdio.h>

int line_count = 0;
int space_count = 0;
int word_count = 0;
int char_count = 0;
int text_count = 0;
%}

%%

\n              { line_count++; char_count++; }
[ \t]           { space_count++; char_count++; }
[A-Za-z]+       { word_count++; text_count++; char_count +=
yyleng; }
[0-9]+          { word_count++; char_count += yyleng; }
[^ \t\nA-Za-z0-9] { char_count += yyleng; }

%%


int main(int argc, char **argv)
```

```
{
    yylex();
    printf("Lines       : %d\n", line_count);
    printf("Spaces      : %d\n", space_count);
    printf("Words       : %d\n", word_count);
    printf("Text tokens: %d\n", text_count);
    printf("Characters : %d\n", char_count);
    return 0;
}

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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./count < input.txt
Lines       : 7
Spaces      : 27
Words       : 26
Text tokens: 24
Characters : 158
```



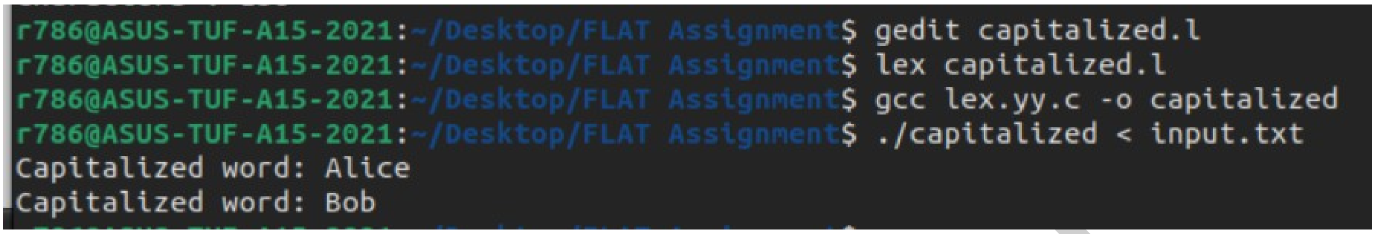
```
Hello, World! This is a sample text.
It includes numbers like 123 and 456.

Lex tools help analyze text - efficiently.

Tabs    and spaces    are counted too.
```

## 2. LEX Program to identify capitalized words from a string.

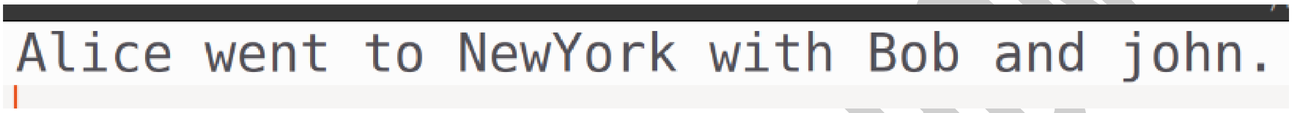
### Program:

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

%%

[A-Z][a-z]+     { printf("Capitalized word: %s\n", yytext); }
[A-Za-z]+       { /* Other words: ignore */ }
[ \t\n]+        { /* Skip whitespace */ }
.               { /* Ignore other characters */ }

%%

int main(int argc, char **argv)
{
    yylex();
    return 0;
}

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

**OUTPUT:**



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit capitalized.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex capitalized.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o capitalized
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./capitalized < input.txt
Capitalized word: Alice
Capitalized word: Bob
```



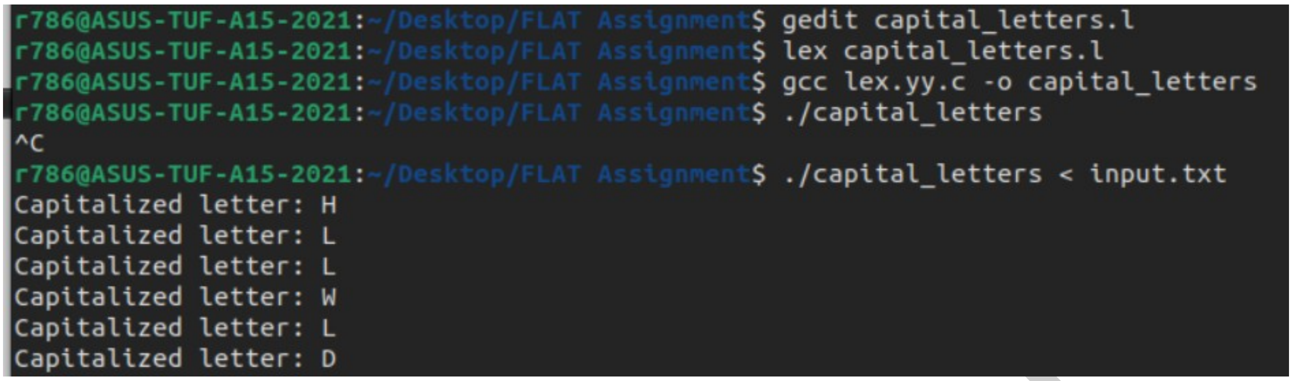
```
Alice went to NewYork with Bob and john.
```

## 3. LEX Program to identify capitalized lettes from a string.

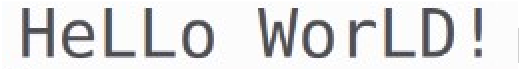
### Program:

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

%%

[A-Z]           { printf("Capitalized letter: %s\n", yytext); }
[a-z]           { /* Lowercase letters - ignore */ }
[ \t\n]+        { /* Whitespace - ignore */ }
.               { /* Other characters - ignore */ }

%%

int main(int argc, char **argv)
{
    yylex();
    return 0;
}

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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit capital_letters.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex capital_letters.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o capital_letters
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./capital_letters
^C
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./capital_letters < input.txt
Capitalized letter: H
Capitalized letter: L
Capitalized letter: L
Capitalized letter: W
Capitalized letter: L
Capitalized letter: D
```



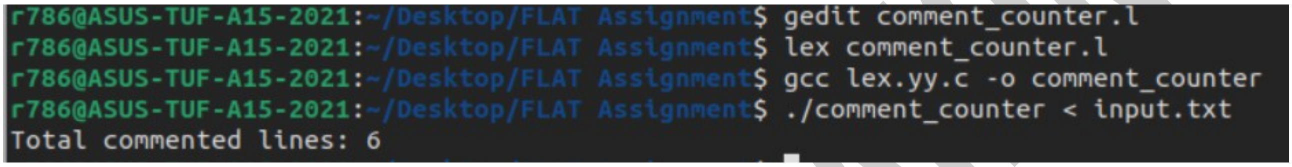
```
HeLLo WorLD!
```

## 4. LEX Program to count number of commented lines.

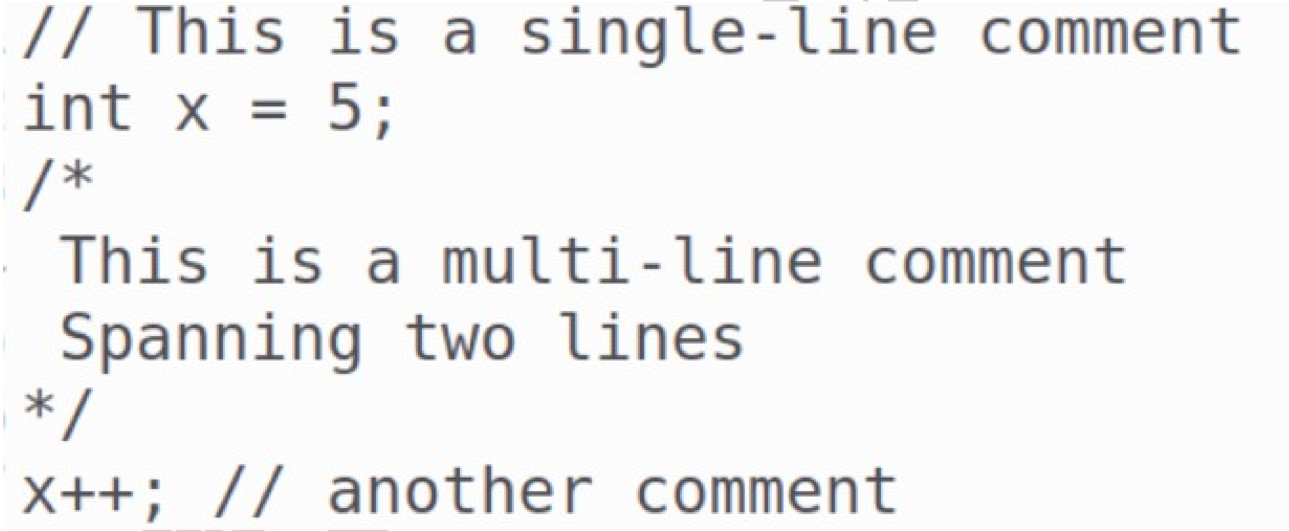
## Program:

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

%%

"//".*                  { comment_line_count++; }
// Single-line comment
"/*"([^*]|\*+[^*/])*\*+"/" {
for (int i = 0; i < yyleng; i++) {
      if (yytext[i] == '\n') comment_line_count++;
     }
comment_line_count++; // At least one line
}
\n          { /* ignore other newlines */ }
.|\t|[a-zA-Z0-9]  { /* ignore other characters */ }



%%

int main(int argc, char **argv)
{
    yylex();
    printf("Total commented lines: %d\n", comment_line_count);
    return 0;
}

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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit comment_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex comment_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o comment_counter
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./comment_counter < input.txt
Total commented lines: 6
```



```
// This is a single-line comment
int x = 5;
/*
 This is a multi-line comment
 Spanning two lines
*/
x++; // another comment
```

## 5. LEX Program to recognize all valid arithmetic sequence.

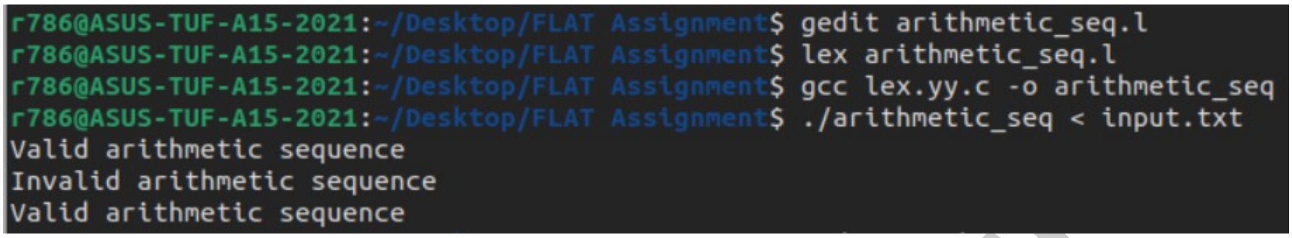
### Program:

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

int prev = 0, diff = 0, count = 0, valid = 1;

void reset() {
    prev = 0;
    diff = 0;
    count = 0;
    valid = 1;
}
%}

%%

[0-9]+ {
    int num = atoi(yytext);
    if (count == 0) {
        prev = num;
    } else if (count == 1) {
        diff = num - prev;
        prev = num;
    } else {
        if (num - prev != diff)
```

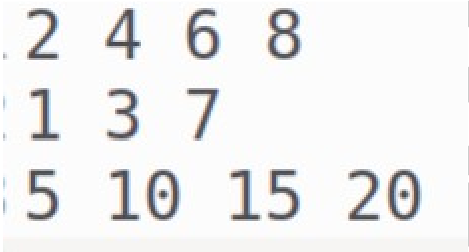
```
            valid = 0;
        prev = num;
    }
    count++;
}

\n {
    if (count > 1) {
        if (valid)
            printf("Valid arithmetic sequence\n");
        else
            printf("Invalid arithmetic sequence\n");
    }
    reset();
}

[ \t]+ ; // Skip spaces/tabs
.      ; // Ignore other characters

%%

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

int yywrap() {
    return 1;
```

```
}
```

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit arithmetic_seq.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex arithmetic_seq.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o arithmetic_seq
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./arithmetic_seq < input.txt
Valid arithmetic sequence
Invalid arithmetic sequence
Valid arithmetic sequence
```



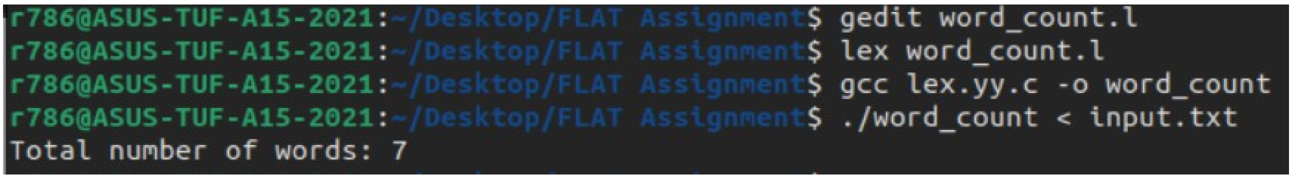
```
2 4 6 8
1 3 7
5 10 15 20
```

# 6. LEX Program to count the number of words.

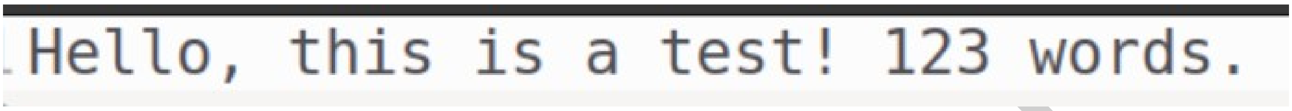
## Program:

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

int word_count = 0;
%}

%%

[A-Za-z0-9]+    { word_count++; }     // Match words (letters or
digits)
[ \t\n]+        ;                     // Skip whitespace
.               ;                     // Ignore other characters

%%

int main() {
    yylex();
    printf("Total number of words: %d\n", word_count);
    return 0;
}

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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit word_count.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex word_count.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o word_count
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./word_count < input.txt
Total number of words: 7
```



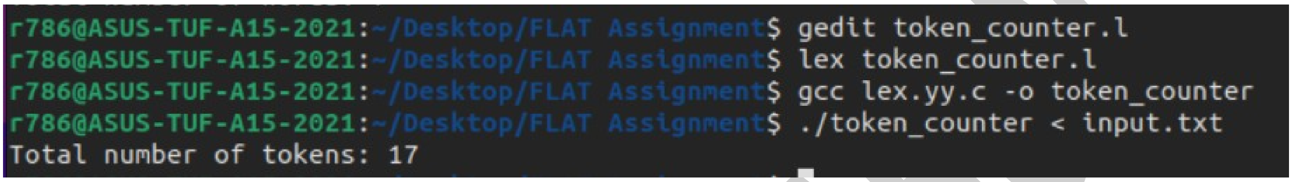
```
Hello, this is a test! 123 words.
```

## 7. LEX Program to count the number of tokens.

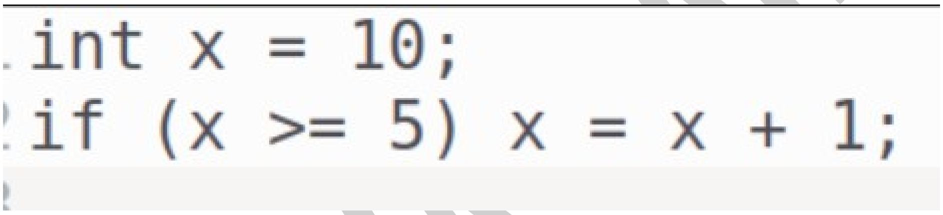
## Program:

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

int token_count = 0;
%}

%%

[0-9]+              { token_count++; }                // Numbers
// Identifiers/words
[A-Za-z_][A-Za-z0-9_]* { token_count++; }
// Multi-char operators
"=="|"!="|"<="|">="  { token_count++; }
// Single-char symbols
[+\-*/=<>;:,(){}]    { token_count++; }
// Ignore whitespace
[ \t\n]+             ;
// Ignore unknowns
.                    ;

%%

int main() {
    yylex();
    printf("Total number of tokens: %d\n", token_count);
    return 0;
}


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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit token_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex token_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o token_counter
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./token_counter < input.txt
Total number of tokens: 17
```



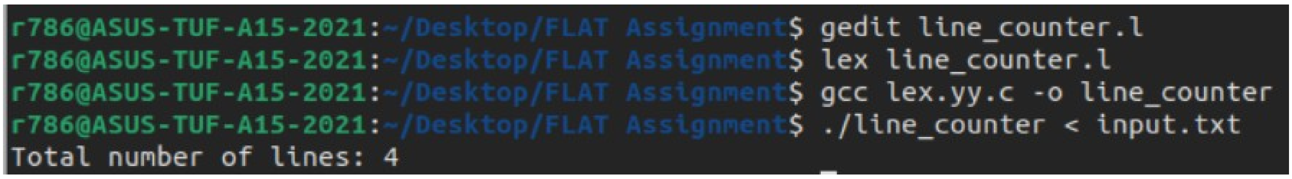
```
int x = 10;
if (x >= 5) x = x + 1;
```

## 8. LEX Program to count no. Of lines in text file.

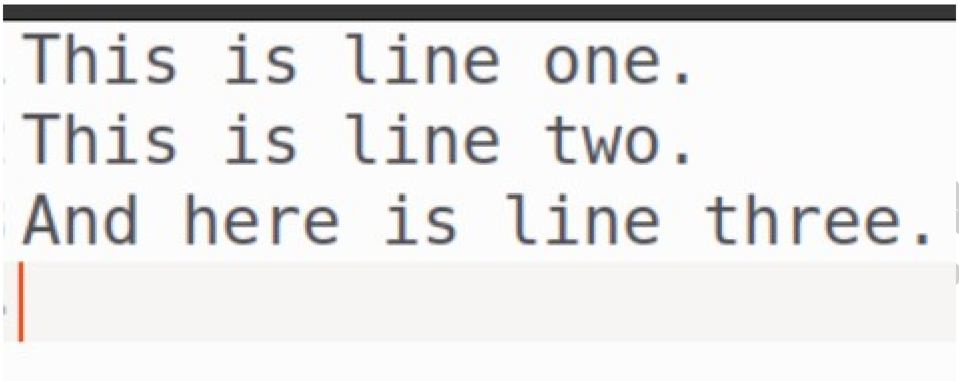
### Program:

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

%%

\n      { line_count++; }      // Count each newline
.       ;                      // Ignore other characters

%%

int main() {
    yylex();
    printf("Total number of lines: %d\n", line_count);
    return 0;
}

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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit line_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex line_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o line_counter
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./line_counter < input.txt
Total number of lines: 4
```



```
This is line one.
This is line two.
And here is line three.

```

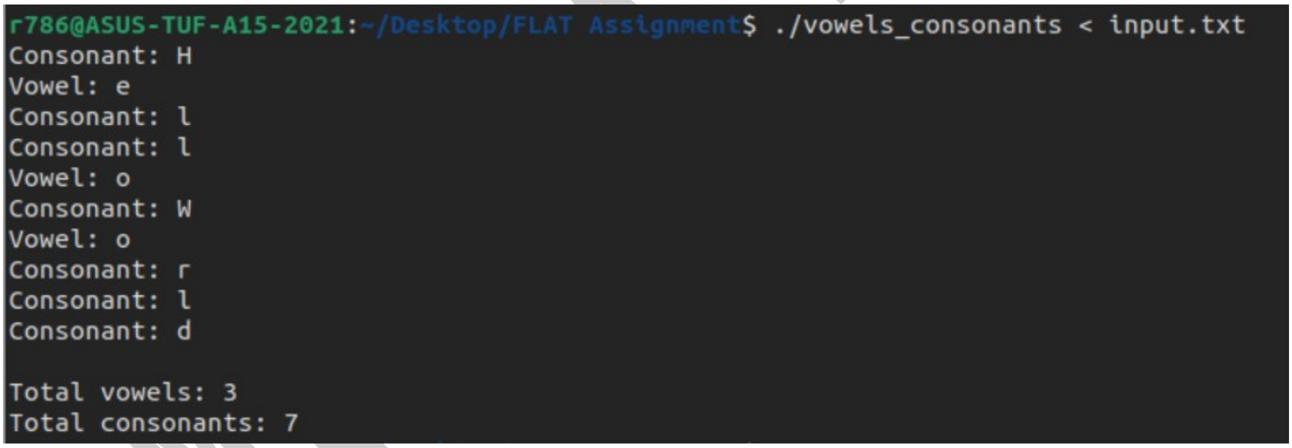
## 9. LEX Program to detect vowel and consonants.

### Program:

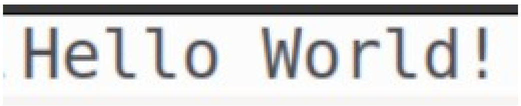
```
%{
#include <stdio.h>

int vowel_count = 0;
int consonant_count = 0;
%}

%%

[aAeEiIoOuU]    {
    printf("Vowel: %s\n", yytext);
    vowel_count++;
}

[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z] {
    printf("Consonant: %s\n", yytext);
    consonant_count++;
}

[ \t\n\r]+      ; // Skip whitespace
.               ; // Ignore other characters (punctuation, digits, etc.)

%%
```

```
int main() {
    yylex();
    printf("\nTotal vowels: %d\n", vowel_count);
    printf("Total consonants: %d\n", consonant_count);
    return 0;
}

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

## OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./vowels_consonants < input.txt
Consonant: H
Vowel: e
Consonant: l
Consonant: l
Vowel: o
Consonant: W
Vowel: o
Consonant: r
Consonant: l
Consonant: d

Total vowels: 3
Total consonants: 7
```



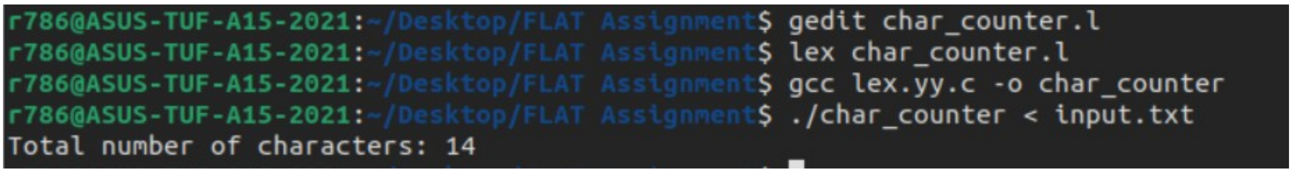
```
Hello World!
```

## 10. LEX Program to count no. of characters in text file.

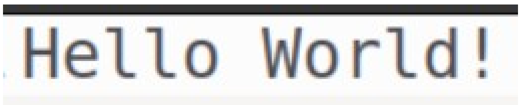
### Program:

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

int char_count = 0;
%}

%%

.       { char_count++; }   // Match any single character (including whitespace)
\n      { char_count++; }   // Count newline separately if needed

%%

int main() {
    yylex();
    printf("Total number of characters: %d\n", char_count);
    return 0;
}

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

### OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit char_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex char_counter.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc lex.yy.c -o char_counter
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./char_counter < input.txt
Total number of characters: 14
```



```
Hello World!
```

# YACC PROGRAMS:

1. Write a YACC Program for the following grammer:

(i) S→L=R|R

L→*R|id|num

R→L

(ii) D→TL

T→int|float

L→L,id|id

## 1. Write a YACC Program for the following grammer:

**(i) S->L=R|R**

**L->*R|id|num**

**R->L**

**YACC Program:**

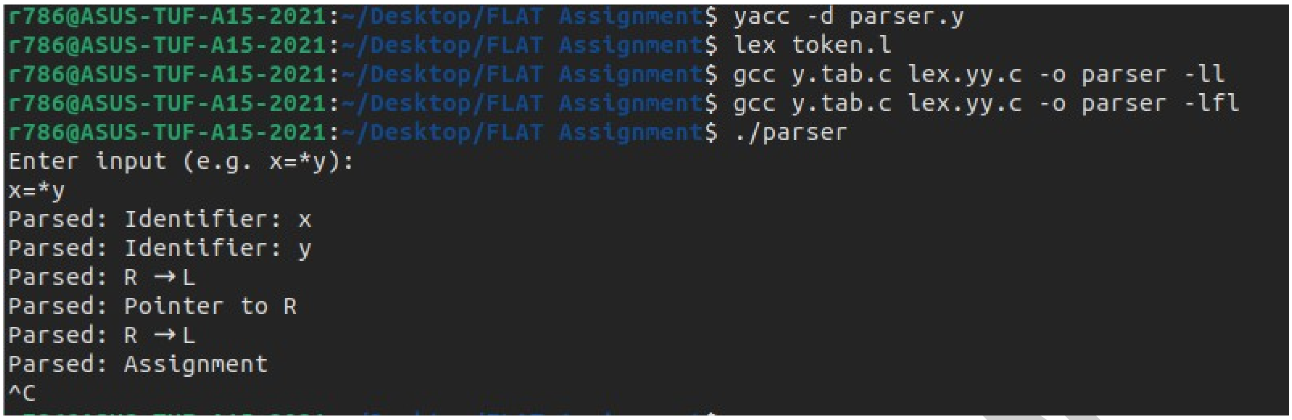
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

%union {
    char* str;
}

%token <str> ID NUM
%token ASSIGN STAR
%type <str> S L R

%%

S : L ASSIGN R { printf("Parsed: Assignment\n"); }
  | R          { printf("Parsed: Expression\n"); }
  ;
```

```
L : STAR R     { printf("Parsed: Pointer to R\n"); }
  | ID         { printf("Parsed: Identifier: %s\n", $1); free($1);
}
  | NUM        { printf("Parsed: Number: %s\n", $1); free($1); }
  ;

R : L          { printf("Parsed: R -> L\n"); }
  ;

%%

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

int main() {
    printf("Enter input (e.g. x=*y):\n");
    return yyparse();
}
```

## LEX Program:

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

%%

"="     { return ASSIGN; }
"*"     { return STAR; }
[0-9]+  { yylval.str = strdup(yytext); return NUM; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return ID; }
[ \t\n] ; // Skip whitespace
.       { return yytext[0]; }

%%

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

# OUTPUT:



```
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ yacc -d parser.y
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex token.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc y.tab.c lex.yy.c -o parser -ll
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc y.tab.c lex.yy.c -o parser -lfl
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./parser
Enter input (e.g. x=*y):
x=*y
Parsed: Identifier: x
Parsed: Identifier: y
Parsed: R →L
Parsed: Pointer to R
Parsed: R →L
Parsed: Assignment
^C
```

(ii) D->TL

T->int|float

L->L,id|id

## YACC Program:

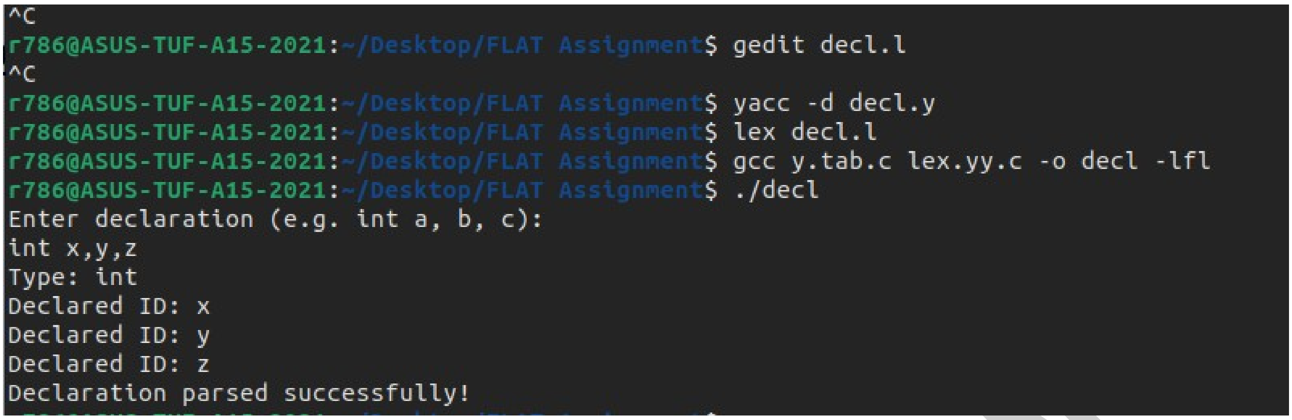
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

%union {
    char* str;
}

%token <str> ID
%token INT FLOAT COMMA
%type <str> D T L

%%

D : T L {
        printf("Declaration parsed successfully!\n");
    }
  ;
```

```
T : INT   { printf("Type: int\n"); }
  | FLOAT { printf("Type: float\n"); }
  ;

L : L COMMA ID {
        printf("Declared ID: %s\n", $3);
        free($3);
    }
  | ID {
        printf("Declared ID: %s\n", $1);
        free($1);
    }
  ;

%%

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

int main() {
    printf("Enter declaration (e.g. int a, b, c):\n");
    return yyparse();
}
```

## LEX Program:

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

%%

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

%%

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

## Output:



```
^C
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gedit decl.l
^C
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ yacc -d decl.y
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ lex decl.l
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ gcc y.tab.c lex.yy.c -o decl -lfl
r786@ASUS-TUF-A15-2021:~/Desktop/FLAT Assignment$ ./decl
Enter declaration (e.g. int a, b, c):
int x,y,z
Type: int
Declared ID: x
Declared ID: y
Declared ID: z
Declaration parsed successfully!
```