# LAB MANUAL

of

# Compiler Design Laboratory

# (CSE606)

## Bachelor of Technology (CSE)

By

**Poojit Luhar (22000407)**

Third Year, Semester 6

*Course In-charge: Prof. Vaibhavi Patel*

Department of Computer Science and Engineering

School Engineering and Technology

Navrachana University, Vadodara

Spring Semester

(2025)

## Question 1

### a)

**AIM:** Write a program to recognize strings starts with ‘a’ over {a, b}.

**PROGRAM CODE:**

```
#include<stdio.h>
//WAP to accept strings starting with a over (a,b)
int main()
{
   char input[10];
   int state = 0, i = 0;
   printf("Enter a string :");
   scanf("%s",input);

   while(input[i] !=  '\0'){
      switch (state)
      {
      case 0:
         if (input[i]=='a'){
            state = 1;
         }

         else if(input[i]=='b'){
            state = 2;
         }
         else{
            state = 3;
         }
         break;
      case 1:
         if (input[i]=='a'||input[i]=='b' ){
            state = 1;
         }

         else{
            state = 3;
         }
```

```
            break;
        case 2:
            if (input[i]=='a'||input[i]=='b' ){
                state = 2;
            }

            else{
                state = 3;
            }
            break;
        case 3:
            if (input[i]=='a'||input[i]=='b' ){
                state = 3;
            }

            else{
                state = 3;
            }
            break;

        default:
            break;
        }
        i++;
    }
    printf("state is %d", state);
    return 0;
}
```

**OUTPUT:**

```
Enter a string :abaa
state is 1%
```

**b)**

**AIM:** Write a program to recognize strings end with ‘a’.

**PROGRAM CODE:**

```
#include<stdio.h>
#include<string.h>
//WAP to accept strings ending with a over (a,b)
int main()
{
   char input[10];
   int state = 0, i = 0;
   printf("Enter a string :");
   scanf("%s",input);

   while(input[i] !=  '\0'){
      switch (state)
      {
      case 0:
         if (input[i]=='a'){
            state = 1;
         }

         else if(input[i]=='b'){
            state = 0;
         }
         else{
            state = 2;
         }
         break;
      case 1:
         if (input[i]=='a' ){
            state = 1;
         }
         else if(input[i]=='b'){
            state = 0;
         }

         else{
```

```
            state = 2;
        }
        break;
    case 2:
        if (input[i]=='a'||input[i]=='b' ){
            state = 2;
        }

        else{
            state = 2;
        }
        break;

    default:
        break;
    }
    i++;
  }
  printf("state is %d", state);
  if (state == 1){
    printf("\nString is accepted");
  }
  else{
    printf("\nString is not accepted");
  }
  return 0;
}
```

**OUTPUT:**

```
Enter a string :aba
state is 1
String is accepted%
```

```
Enter a string :abb
state is 0
String is not accepted%
```

**c)**

**AIM:** Write a program to recognize strings end with ‘ab’. Take the input from text file.

**PROGRAM CODE:**

```
#include<stdio.h>
/*Write a program to recognize strings end with ‘ab’.
Take the input from text file.*/
int main()
{
   char input[10];
   int state = 0, i = 0;
   FILE *filePointer = NULL;
   filePointer = fopen("testFile3.txt", "r");
   // check there is no error opening the file
   if (filePointer == NULL)
   {
      perror("Error: ");
      return(-1);
   }
   if(fgets(input, 60, filePointer) != NULL) {
      // print the return value (aka string read in) to terminal
      printf("%s", input);
   }
   fclose(filePointer);
   filePointer = NULL;
   while(input[i] !=  '\0'){
      switch (state)
      {
      case 0:
         if (input[i]=='a'){
            state = 1;
         }
         else{
            state = 0;
         }
         break;
      case 1:
         if (input[i]=='a'){
```

```
            state = 1;
        }
        else if (input[i]=='b') {
            state = 2;
        }
        else{
            state = 0;
        }
        break;

    case 2:
        if (input[i]=='a'){
            state = 1;
        }
        else{
            state = 0;
        }
        break;
    default:
        break;
    }
    i++;
}
    printf("\nstate is %d \n", state);
    if (state == 2){
        printf("String is accepted");
    }
    else{
        printf("String is not accepted");
    }
    return 0;
}
```

**OUTPUT:**

```
abc
state is 0
String is not accepted%
```

```
ccccab
state is 2
String is accepted%
```

**d)**

**AIM:** Write a program to recognize strings contains ‘ab’. Take the input from text file.

**PROGRAM CODE:**

```
#include <stdio.h>
int main() {
    char input[100];
    int state = 0, i = 0;
    FILE *filePointer = NULL;
    filePointer = fopen("testFile4.txt", "r");
    if (filePointer == NULL) {
        perror("Error: ");
        return -1;
    }
    if (fgets(input, sizeof(input), filePointer) != NULL) {
        printf("Input read: %s", input);
    }
    fclose(filePointer);
    while (input[i] != '\0' && input[i] != '\n') {
        switch (state) {
            case 0:
                if (input[i] == 'a') {
                    state = 1;
                }
                break;
            case 1:
                if (input[i] == 'b') {
                    state = 2;
                } else if (input[i] == 'a') {

                    state = 1;
                } else {
                    state = 0;
                }
                break;
            case 2:
                break;
            default:
```

```
            break;
        }
        if (state == 2) {
            break;
        }
        i++;
    }
    if (state == 2) {
        printf("\nString is accepted: contains 'ab'\n");
    } else {
        printf("\nString is not accepted: does not contain 'ab'\n");
    }
    return 0;
}
```

**OUTPUT:**

```
Input read: ccccab
String is accepted: contains 'ab'
```

```
Input read: cccca
String is not accepted: does not contain 'ab'
```

## Question 2

### a)

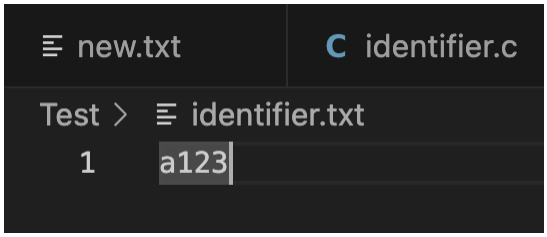
**AIM:** Write a program to recognize the valid identifiers.

**PROGRAM CODE:**

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
char* keywords[] = {
    "auto", "break", "case", "char", "const", "continue", "default",
    "do", "double", "else", "enum", "extern", "float", "for", "goto",
    "if", "int", "long", "register", "return", "short", "signed",
    "sizeof", "static", "struct", "switch", "typedef", "union",
    "unsigned", "void", "volatile", "while"
};
int isKeyword(char *word) {
    for (int i = 0; i < 32; i++) {
        if (strcmp(word, keywords[i]) == 0)
            return 1;
    }
    return 0;
}
int isValidIdentifier(char *str) {
    int i = 0;
    if (!(isalpha(str[0]) || str[0] == '_'))
        return 0;
    for (i = 1; str[i] != '\0'; i++) {
        if (!(isalnum(str[i]) || str[i] == '_'))
            return 0;
    }
    if (isKeyword(str))
        return 0;
    return 1;
}
int main() {
    char input[100];
    FILE *file = fopen("identifier.txt", "r");
```

```
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    fscanf(file, "%s", input);
    fclose(file);
    if (isValidIdentifier(input)) {
        printf("String is a valid identifier\n");
    } else {
        printf("String is not a valid identifier\n");
    }
    return 0;
}
```

**INPUT:**




**OUTPUT:**

**b)**

**AIM:** Write a program to recognize the valid operators.

**PROGRAM CODE:**

```
#include <stdio.h>

int main(){
        char input[100];
        int state = 0, i = 0;

        FILE *file = fopen("operator.txt", "r");
   if (file == NULL) {
      printf("Error opening file.\n");
      return 1;
   }

   fscanf(file, "%s", input);
   fclose(file);

   while(input[i] != '\0'){
        switch(state){
                case 0:
                        if(input[i] == '+'){
                                state = 1;
                                }
                                else if(input[i] == '-'){
                                        state = 5;
                                }
                                else if(input[i] == '*'){
                                        state = 9;
                                }
                                else if(input[i] == '/'){
                                        state = 12;
                                }
                                else if(input[i] == '%'){
                                        state = 15;
                                }
                                else if(input[i] == '&'){
```

```
                state = 18;
        }
        else if(input[i] == '|'){
                state = 21;
        }
        else if(input[i] == '<'){
                state = 24;
        }
        else if(input[i] == '>'){
                state = 28;
        }
        else if(input[i] == '!'){
                state = 32;
        }
        else if(input[i] == '~'){
                state = 34;
        }
        else if(input[i] == '^'){
                state = 35;
        }
        else if(input[i] == '='){
                state = 36;
        }
        break;

case 1:
        if(input[i] == '+'){
                state = 2;
                printf("++,unari operator");
        }
        else if(input[i] == '='){
                state = 3;
                printf("+=,,assignment operator");
        }
        else{
                state = 4;
                printf("+,arithmetic operator");
        }
        break;
```

```
case 5:
        if(input[i] == '-'){
                state = 6;
                printf("--,unari operator");
        }
        else if(input[i] == '='){
                state = 7;
                printf("-=,assignment operator");
        }
        else{
                state = 8;
                printf("+,arithmetic operator");
        }
        break;

case 9:
        if(input[i] == '='){
                state = 10;
                printf("*=,assignment operator");
        }
        else{
                state = 11;
                printf("*,arithmetic operator");
        }
        break;

case 12:
        if(input[i] == '='){
                state = 13;
                printf("/=,assignment operator");
        }
        else{
                state = 14;
                printf("/,arithmetic operator");
        }
        break;

case 15:
        if(input[i] == '='){
                state = 16;
```

```
                printf("%=,assignment operator");
        }
        else{
                state = 17;
                printf("%,arithmetic operator");
        }
        break;

case 18:
        if(input[i] == '&'){
                state = 19;
                printf("&&,Logical operator");
        }
        else{
                state = 20;
                printf("%,Bitwise operator");
        }
        break;

case 21:
        if(input[i] == '|'){
                state =22;
                printf("||,Logical operator");
        }
        else{
                state = 23;
                printf("|,Bitwise operator");
        }
        break;

case 24:
        if(input[i] == '<'){
                state =25;
                printf("<<,Bitwise operator");
        }
        else if(input[i] == '='){
                state =27;
                printf("<=,Relational operator");
        }
        else{
```

```
                        state = 26;
                        printf("< ,Relational operator");
                }
                break;

        case 28:
                if(input[i] == '>'){
                        state =29;
                        printf(">>,Bitwise operator");
                }
                else if(input[i] == '='){
                        state =30;
                        printf(">=,Relational operator");
                }
                else{
                        state = 31;
                        printf("> ,Relational operator");
                }
                break;

        case 32:
                if(input[i] == '='){
                        state =33;
                        printf("!=,Assignment operator");
                }
                break;

        case 36:
                if(input[i] == '='){
                        state =37;
                        printf("==,Relational operator");
                }
                break;

        default:
                break;
        }

i++;
}
```

```
        printf("\nState is %d\n",state);
        if(state == 1){printf("+,arithmetic operator\n");}
        else if(state == 5){printf("-,arithmetic operator\n");}
        else if(state == 9){printf("*,arithmetic operator\n");}
        else if(state == 12){printf("/,arithmetic operator\n");}
        else if(state == 15){printf("%,arithmetic operator\n");}
        else if(state == 18){printf("&,Bitwise operator\n");}
        else if(state == 21){printf("|,Bitwise operator\n");}
        else if(state == 24){printf("<,Relational operator\n");}
        else if(state == 28){printf(">,Relational operator\n");}
        else if(state == 32){printf("!,Logical operator\n");}
        else if(state == 34){printf("~,Bitwise operator\n");}
        else if(state == 35){printf("^,Bitwise operator\n");}
        else if(state == 36){printf("=,Assignment operator\n");}
        return 0;
}
```

**INPUT:**

```
new.txt        C identifi
Test > operator.txt
  1    +
```

**OUTPUT:**

```
State is 1
+,arithmetic operator
```

**c)**

**AIM:** Write a program to recognize the valid number.

**PROGRAM CODE:**

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char input[100];
    int state = 0, i = 0, hasDecimal = 0, hasExponent = 0;
    FILE *file = fopen("input.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fscanf(file, "%s", input);
    fclose(file);
    while (input[i] != '\0') {
        switch (state) {
            case 0:
                if (isdigit(input[i])) {
                    state = 1;
                } else if (input[i] == '+' || input[i] == '-') {
                    state = 2;
                } else {
                    state = 5; // Invalid state
                }
                break;
            case 1:
                if (isdigit(input[i])) {
                    state = 1;
                } else if (input[i] == '.' && hasDecimal == 0) {
                    state = 3;
                    hasDecimal = 1;
                } else if ((input[i] == 'e' || input[i] == 'E') && hasExponent == 0) {
                    state = 4;
                    hasExponent = 1;
                } else {
                    state = 5;
```

```
            }
            break;
        case 2:
            if (isdigit(input[i])) {
                state = 1;
            } else {
                state = 5;
            }
            break;
        case 3:
            if (isdigit(input[i])) {
                state = 3;
            } else if ((input[i] == 'e' || input[i] == 'E') && hasExponent == 0) {
                state = 4;
                hasExponent = 1;
            } else {
                state = 5;
            }
            break;
        case 4:
            if (isdigit(input[i])) {
                state = 4;
            } else if ((input[i] == '+' || input[i] == '-') && (input[i - 1] == 'e' || input[i - 1] == 'E')) {
                state = 4;
            } else {
                state = 5;
            }
            break;
        case 5:
            state = 5;
            break;
        default:
            break;
    }
    i++;
}
printf("State is: %d\n", state);
if (state == 1 || state == 3 || state == 4) {
    printf("It is a Valid number\n");
} else {
```

```
        printf("It is an Invalid number\n");
    }
    return 0;
}
```

**INPUT:**

lab10.txt myfil

Test > input.txt

```
5.67e3
```

**OUTPUT:**

```
State is: 4
It is a Valid number
```

**d)**

**AIM:** Write a program to recognize the valid comments.

**PROGRAM CODE:**

```
#include <stdio.h>
int main(){
    char input[100];
    int state = 0, i = 0;

    FILE *file = fopen("comment.txt", "r");
    if (file == NULL){
        printf("Error Opening file\n");
        return 1;
    }

    fscanf(file, "%s", input);
    fclose(file);

    while(input[i] != '\0'){
        switch(state){
            case 0:
                if(input[i] == '/'){
                    state =1;
                }
                else{
                    state =3;
                }
                break;

            case 1:
                if(input[i] == '/'){
                    state = 2;
                }
                else if(input[i] == '*'){
                    state = 4;
                }
                else{
                    state =3;
```

```
            }
            break;

        case 2:
            if(input[i] != '\0'){
                state =2;
            }
            break;

        case 3:
            state =3;
            break;

        case 4:
            if(input[i] == '*'){
                state = 5;
            }
            else{
                state =4;
            }
            break;

        case 5:
            if(input[i] == '/'){
                state = 6;
            }
            else{
                state =4;
            }
            break;

        case 6:
            state = 3;
            break;

        default:
            break;
    }
    i++;
}
```

```
    printf("State is : %d\n", state);
    if(state == 2 || state == 6){
        printf("It is Valid Comment\n");
    }
    else{
        printf("It is not Valid Comment\n");
        return 0;
    }
}
```

**INPUT:**

lab10.txt | myfile.txt

Test > input.txt

```
1   //This is a comment
2
```

**OUTPUT:**

```
State is : 2
It is Valid Comment
```

**e)**

**AIM:** Program to implement Lexical Analyzer.

**PROGRAM CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define BUFFER_SIZE 1000
void check(char *lexeme);
void processSymbol(char c);
void main() {
    FILE *f1;
    char buffer[BUFFER_SIZE], lexeme[50];
    char c;
    int f = 0, state = 0, i = 0;

    f1 = fopen("input.txt", "r");
    if (f1 == NULL) {
        printf("Error opening file!\n");
        return;
    }

    fread(buffer, sizeof(char), BUFFER_SIZE - 1, f1);
    buffer[BUFFER_SIZE - 1] = '\0';
    fclose(f1);
    while (buffer[f] != '\0') {
        switch (state) {
            case 0:
                c = buffer[f];
                if (isalpha(c) || c == '_') {
                    state = 1;
                    lexeme[i++] = c;
                } else if (isdigit(c)) {
                    state = 2;
                    lexeme[i++] = c;
                } else if (c == '/') {
                    state = 3;
```

```
    } else if (c == ' ' || c == '\t' || c == '\n') {
        state = 0;
    } else {
        processSymbol(c);
        state = 0;
    }
    break;

case 1:
    c = buffer[f];
    if (isalnum(c) || c == '_') {
        lexeme[i++] = c;
    } else {
        lexeme[i] = '\0';
        check(lexeme);
        i = 0;
        state = 0;
        f--;
    }
    break;

case 2:
    c = buffer[f];
    if (isdigit(c)) {
        lexeme[i++] = c;
    } else if (c == '.') {
        state = 4;
        lexeme[i++] = c;
    } else {
        lexeme[i] = '\0';
        printf("%s is a valid number\n", lexeme);
        i = 0;
        state = 0;
        f--;
    }
    break;

case 3:
    c = buffer[f];
    if (c == '/') {
```

```
                while (buffer[f] != '\n' && buffer[f] != '\0') {
                    f++;
                }
            } else if (c == '*') {
                f++;
                while (buffer[f] != '\0' && !(buffer[f] == '*' && buffer[f + 1] == '/')) {
                    f++;
                }
                f += 2;
            } else {
                printf("/ is a symbol\n");
                f--;
            }
            state = 0;
            break;

        case 4:
            c = buffer[f];
            if (isdigit(c)) {
                lexeme[i++] = c;
            } else {
                lexeme[i] = '\0';
                printf("%s is a valid float number\n", lexeme);
                i = 0;
                state = 0;
                f--;
            }
            break;

        default:
            state = 0;
            break;
        }
        f++;
    }
}
void check(char *lexeme) {
    char *keywords[] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do",
        "double", "else", "enum", "extern", "float", "for", "goto", "if",
```

```
        "inline", "int", "long", "register", "restrict", "return", "short", "signed",
        "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void", "volatile",
"while"
    };

    for (int i = 0; i < 32; i++) {
        if (strcmp(lexeme, keywords[i]) == 0) {
            printf("%s is a keyword\n", lexeme);
            return;
        }
    }
    printf("%s is an identifier\n", lexeme);
}
void processSymbol(char c) {
    char symbols[] = {';', ',', '{', '}', '(', ')', '[', ']', '+', '-', '*', '=', '<', '>', '!'};
    int symbolCount = sizeof(symbols) / sizeof(symbols[0]);

    for (int i = 0; i < symbolCount; i++) {
        if (c == symbols[i]) {
            printf("%c is a symbol\n", c);
            return;
        }
    }
}
```

**INPUT:**

identifier.c | identifier.txt | parse.c

Test > input.txt

```
int a = 10;
float pi = 3.14;
char c = 'x';
if (a > 5) {
    // This is a comment
    a = a + 1;
    /* This is a
       multiline comment */
}
```

**OUTPUT:**

```
int is a keyword
a is an identifier
= is a symbol
10 is a valid number
; is a symbol
float is a keyword
pi is an identifier
= is a symbol
3.14 is a valid float number
; is a symbol
char is a keyword
c is an identifier
= is a symbol
x is an identifier
; is a symbol
if is a keyword
( is a symbol
a is an identifier
> is a symbol
5 is a valid number
) is a symbol
{ is a symbol
a is an identifier
= is a symbol
a is an identifier
+ is a symbol
1 is a valid number
; is a symbol
} is a symbol
```

## Question 4

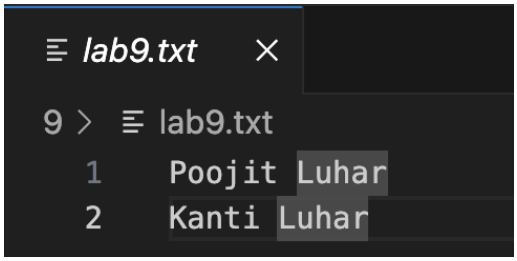
### a)

**AIM:** Write a Lex program to take input from text file and count no of characters, no. of lines & no. of words.

**PROGRAM CODE:**

```
%{
#include<stdio.h>
int characters=0;
int words=0;
int lines=1;
%}
%%
[a-zA-z]    {characters++;}
" " {words++;}
\n  {lines++;words++;}
.   {characters++;}
%%
int main(){
yyin=fopen("lab9.txt","r");
yylex();
printf("This file is containing %d characters\n",characters);
printf("This file is containing %d words\n",words);
printf("This file is containing %d lines\n",lines);
}
int yywrap(){return(1);}
```

**INPUT:**



**OUTPUT:**

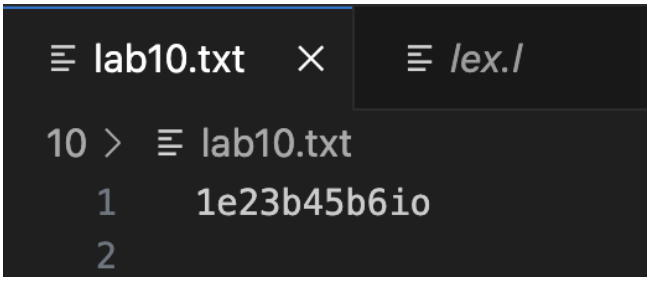
```
(base) poojitluhar@Poojits-MacBook-Pro 9 % ./lex
This file is containing 21 characters
This file is containing 3 words
This file is containing 2 lines
```

**b)**

**AIM:** Write a Lex program to take input from text file and count number of vowels and consonants.

**PROGRAM CODE:**

```
%{
#include<stdio.h>
int consonants=0;
int vowels=0;
%}
%%
[aeiouAEIOU] {vowels++;}
[a-zA-Z] {consonants++;}
. ;
%%
int main(){
yyin=fopen("lab10.txt","r");
yylex();
printf("This file is containing %d vowels\n",vowels);
printf("This file is containing %d consonants\n",consonants);
}
int yywrap(){return(1);}
```




**INPUT:**

**OUTPUT:**

```
This file is containing 3 vowels
This file is containing 2 consonants
```

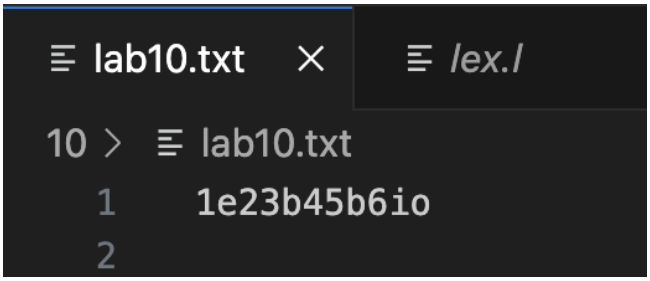
**c)**

**AIM:** Write a Lex program to print out all numbers from the given file.

**PROGRAM CODE:**

```
%{
#include<stdio.h>
int numberCount=0;
%}
%%
[0-9]+(\.[0-9]+)? {numberCount++;}
. ;
%%
int main(){
yyin=fopen("lab10.txt","r");
yylex();
printf("This file is containing %d numbers\n",numberCount);
}
int yywrap(){return(1);}
```

**INPUT:**



**OUTPUT:**

```
(base) poojitluhar@Poojits-MacBook-Pro 10 % ./lex2

This file is containing 6 numbers
```

**d)**

**AIM:** Write a Lex program which adds line numbers to the given file and display the same into different file.

**PROGRAM CODE:**

```
%{
#include<stdio.h>
int line_number=1;
%}
%%
.+ {fprintf(yyout,"%d: %s", line_number,yytext);line_number++;}
. ;
%%
int main(){
yyin=fopen("myfile.txt","r");
yyout=fopen("op.txt", "w");
yylex();
printf("done");
return 0;
}
int yywrap(){return(1);}
```

**INPUT:**

**OUTPUT:**

lab10.txt  op.txt

10 > op.txt

```
1: <h1>hello</h1>
2: <!--helloji-->
```

**e)**

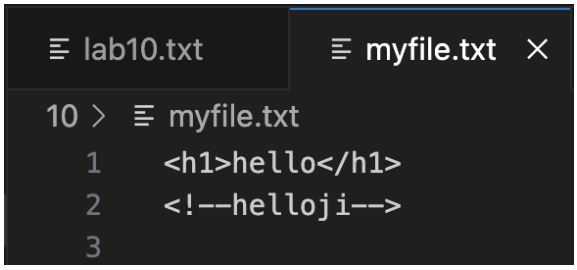
**AIM:** Write a Lex program to printout all markup tags and HTML comments in file.

**PROGRAM CODE:**

```
%{
#include<stdio.h>
int num=0;
%}
%%
"<"[A-Za-z0-9]+">" printf("%s is a valid markup tag\n", yytext);
"<!--"(.|\n)*"-->" num++;
\n ;
. ;
%%

int main(){
yyin=fopen("myfile.txt","r");
yylex();
printf("%d comment", num);
return 0;
}
int yywrap(){return(1);}
```

**INPUT:**



**OUTPUT:**

```
<h1> is a valid markup tag
1 comment%
```

## Question 5

### a)

**AIM:** Write a Lex program to count the number of C comment lines from a given C program. Also eliminate them and copy that program into separate file.

**PROGRAM CODE:**

```
%{
#include <stdio.h>
int c = 0;  // Counter for comments
%}

%%
"/*"([^*]|\*+[^/]|[\n])*"*"+"/"   { fprintf(yyout, ""); c++; }  // Multiline comment
"//".*            { fprintf(yyout, " "); c++; }  // Single-line comment
.                 { fprintf(yyout, "%s", yytext); }  // Copy other text to output
%%

int main() {
   yyin = fopen("code.txt", "r");    // Open input file
   yyout = fopen("output.txt", "w"); // Open output file
   yylex();  // Perform lexical analysis
   printf("%d comments\n", c);  // Print the number of comments
   fclose(yyin);  // Close input file
   fclose(yyout); // Close output file
   return 0;
}

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

**INPUT:**

lab10.txt | myfile.txt | code.txt

code.txt

```
// This is a single-line comment

int main() {
    printf("Hello World!\n"); /* Inline multi-line comment */
    return 0;
}

/*
  This is
  a block comment
*/
```

**OUTPUT:**

lab10.txt | myfile.txt | output.txt

output.txt

```


int main() {
    printf("Hello World!\n");
    return 0;
}

```

**b)**

**AIM:** Write a Lex program to recognize keywords, identifiers, operators, numbers, special symbols, literals from a given C program.

**PROGRAM CODE:**

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

void print_token(const char* type, const char* text) {
   printf("<%-15s>  %s\n", type, text);
}
%}

%%
"int"|"float"|"char"|"if"|"else"|"while"|"for"|"return"|"void"|"double" {
   print_token("Keyword", yytext);
}

[0-9]+"."[0-9]+  { print_token("Float", yytext); }
[0-9]+          { print_token("Integer", yytext); }

"=="|"!="|"<="|">="|"++"|"--"     { print_token("Operator", yytext); }
"+"|"-"|"*"|"/"|"="|"<"|">"       { print_token("Operator", yytext); }

[a-zA-Z_][a-zA-Z0-9_]*            { print_token("Identifier", yytext); }

\"([^\\\"]|\\.)*\"                { print_token("String Literal", yytext); }
\'([^\\\']|\\.)\'                 { print_token("Char Literal", yytext); }

[\(\){};,]                        { print_token("Special Symbol", yytext); }

[ \t\n]                           { /* Ignore whitespace */ }

.                                 { print_token("Unknown", yytext); }
%%
```

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

int main() {
    printf("Enter a C program snippet (Ctrl+D to end):\n");
    yylex();
    return 0;
}
```

**OUTPUT:**

```
Enter a C program snippet (Ctrl+D to end):
int x = 10;
float y = 2.5;
if (x > y) {
    printf<Keyword        >  int
<Identifier     >  x
<Operator       >  =
<Integer        >  10
<Special Symbol >  ;
<Keyword        >  float
<Identifier     >  y
<Operator       >  =
<Float          >  2.5
<Special Symbol >  ;
<Keyword        >  if
<Special Symbol >  (
<Identifier     >  x
<Operator       >  >
<Identifier     >  y
<Special Symbol >  )
<Special Symbol >  {
("x is greater");
}<Identifier     >  printf
<Special Symbol >  (
<String Literal >  "x is greater"
<Special Symbol >  )
<Special Symbol >  ;
```

## Question 6

**AIM:** Program to implement Recursive Descent Parsing in C.

**PROGRAM CODE:**

```
#include<stdio.h>
#include<stdlib.h>
char s[20];
int i = 1;
char l;
int match(char l);
int E1();
int E(){
   if(l == 'i'){
      match('i');
      E1();
   }
   else{
      printf("Error Parsing Tree");
      exit(1);
   }
}
int E1(){
   if(l == '+'){
      match('+');
      match('i');
      E1();
   }
   else{
      return 0;
   }
}
int match(char t){
   if(l == t){
      l = s[i];
      i++;
   }
   else{
      printf("Syntax Error");
```

```
        exit(1);
    }
    return 0;
}
int main(){
    printf("Enter a String: ");
    scanf("%s", &s);
    l = s[0];
    E();
    if(l == '$'){
        printf("Parsing Sussesfull");
    }
    else{
        printf("Error Parsing Tree");
    }
}
```

**OUTPUT:**

```
Enter a String: i+i+i+i$
Parsing Sussesfull%
```

## Question 7

## b)

**AIM:** Create Yacc and Lex specification files to recognizes arithmetic expressions involving +, -, * and / .

**PROGRAM CODE:**

**yacc.y**

```
%{
#include <stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token NUM
%token id
%%
S: E '\n' { printf("valid synatx"); return(0); }
E: E '+' T {  }
 | E '-' T {  }
 | T       {  }
T: T '*' F {  }
 | F       {  }
F:NUM   {  }
  | id   {  }
%%
void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
}
int main() {
    yyparse();
    return 0;
}
```

**lex.l**

```
%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUM;}
[a-zA-Z_][a-zA-Z_0-9]* {return id;}
[-+*\n] {return *yytext;}
[ \t] { }
. yyerror("invalid character");
%%
int yywrap() {
return 0;
}
```

**OUTPUT:**

```
x + 3 * y
valid synatx%
```

**c)**

**AIM:** Create Yacc and Lex specification files are used to generate a calculator which accepts integer type arguments.

**PROGRAM CODE:**

**INPUT:**

**lex.l**

```
%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUM;}
[-+*\n] {return *yytext;}
[ \t] { }
. yyerror("invalid character");
%%
int yywrap() {
 return 0;
}
```

**yacc.y**

```
%{
 #include <stdio.h>
 int yylex(void);
 void yyerror(char *);
%}
%token NUM
%%
S: E '\n' { printf("%d\n", $1); return(0); }
E: E '+' T    { $$ = $1 + $3; }
 | E '-' T   { $$ = $1 - $3; }
 | T         { $$ = $1; }
```

```
T : T '*' F { $$ = $1 * $3; }
 | F        { $$ = $1; }
F:NUM    { $$ = $1; }
%%
void yyerror(char *s) {
 fprintf(stderr, "%s\n", s);
}
int main() {
 yyparse();
 return 0;
}
```

**OUTPUT:**

```
2 + 3 * 4
14
```

## d)

**AIM:** Create Yacc and Lex specification files are used to convert infix expression to postfix expression.

**PROGRAM CODE:**

**yacc.y**

```
%{
 #include <stdio.h>
 int yylex(void);
 void yyerror(char *);
%}
%union {
    char *str;
    int num;
}
%token <num> INTEGER
%token <str> ID
%%
S: E '\n'       { printf("\n"); }
E: E '+' T      { printf("+"); }
 | E '-' T      { printf("-"); }
 | T            { }
T : T '*' F     { printf("*"); }
 | F            { }
F:  INTEGER     { printf("%d",$1); }
 | ID            { printf("%s",$1); }
%%
void yyerror(char *s) {
 fprintf(stderr, "%s\n", s);
}
int main() {
 yyparse();
 return 0;
}
```

**lex.l**

```
%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc.tab.h"
%}
%%
[0-9]+ {yylval.num = atoi(yytext); return INTEGER;}
[a-zA-Z_][a-zA-Z0-9_]* {yylval.str = yytext; return ID;}
[-+*\n] {return *yytext;}
[ \t] { }
. yyerror("invalid character");
%%
int yywrap() {
 return 0;
}
```

**OUTPUT:**

```
x + 3 * y
x3y*+
```