Lab Manual

of

Compiler Design Laboratory

(CSE606)

Bachelor of Technology (CSE)

By

Ramoliya Kaushal (22000409)

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)

INDEX

PRACTICAL: - 1

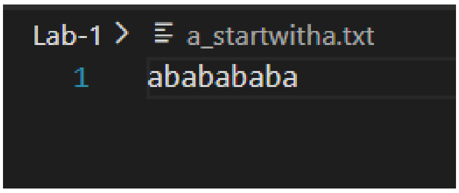
AIM:

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

PROGRAM CODE: -

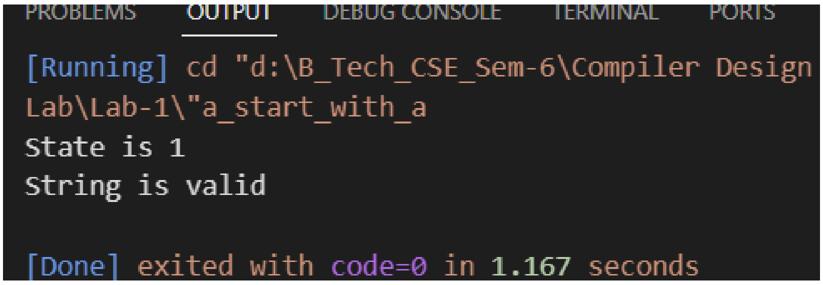
```
#include <stdio.h>


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


    FILE *file = fopen("a_startwitha.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] == '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:
                state = 3;
                break;
            default:
                break;
        }
        i++;
    }

    printf("State is %d\n", state);
    if (state == 1) {
        printf("String is valid\n");
    } else {
        printf("String is Invalid\n");
    }

    return 0;
}
```

INPUT: -



```
ababababa
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Design
Lab\Lab-1\"a_start_with_a
State is 1
String is valid

[Done] exited with code=0 in 1.167 seconds
```

AIM:

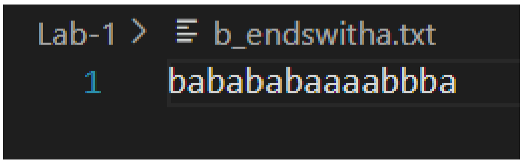
b). Write a program to recognize strings end with ‘a’.

PROGRAM CODE: -

```
#include <stdio.h>

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

    FILE *file = fopen("b_endswitha.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] == '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:
                state = 2;
                break;

            default:
                break;
        }
        i++;
    }

    printf("State is %d\n", state);

    if (state == 1) {
        printf("String is valid\n");
    } else {
        printf("String is Invalid\n");
    }
```

```
        return 0;
    }
```

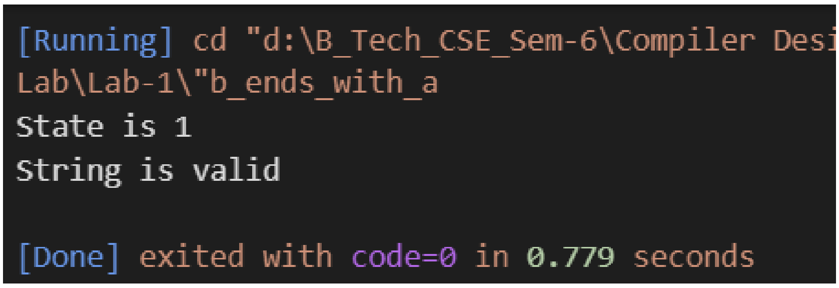
INPUT: -



Lab-1 > b_endswitha.txt

```
babababaaaabbba
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Desi
Lab\Lab-1\"b_ends_with_a
State is 1
String is valid

[Done] exited with code=0 in 0.779 seconds
```

AIM:

c). Write a program to recognize strings end with ‘ab’. Take the input from text file.

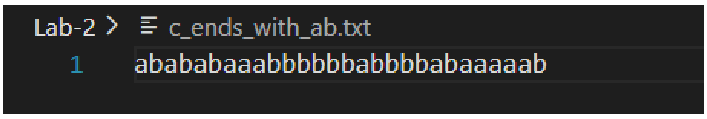
PROGRAM CODE: -

```
#include <stdio.h>

int main () {

    char input [100];

    int state = 0, i = 0;


    FILE *file = fopen("c_ends_with_ab.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] == 'a’) {

                    state = 1;

                }

                else if(input[i] == 'b’) {

                    state = 0;

                }

                else {

                    state = 3;

                }
```

```
 break;

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

case 2:
 if(input[i] == 'a’) {
  state = 1;
 }
 else if(input[i] == 'b’) {
  state = 0;
 }
 else {
  state = 3;
 }
 break;

case 3:
 state = 3;
 break;
```

```
            default:
                break;
        }

        i++;
    }

    printf("State is %d\n",state);

    if(state == 2 ){
        printf("Stering is velid\n");
    }
    else{
        printf("Stering is Invelid\n");
    }



    return 0;
}
```

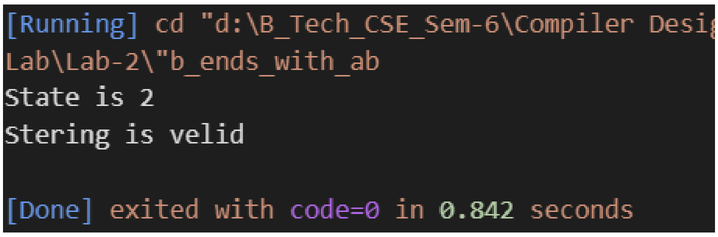
INPUT: -



Lab-2 > c_ends_with_ab.txt

```
abababaaabbbbbbabbbbabaaaaab
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Desi
Lab\Lab-2\"b_ends_with_ab
State is 2
Stering is velid

[Done] exited with code=0 in 0.842 seconds
```

AIM:

d). Write a program to recognize strings contains ‘ab’. Take the input from text file.

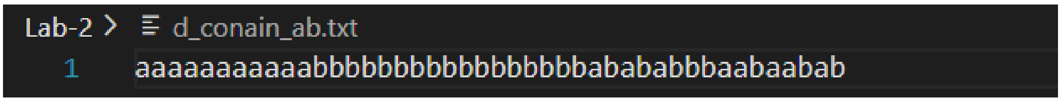
PROGRAM CODE: -

```
#include <stdio.h>

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

   FILE *file = fopen("d_conain_ab.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] == 'a'){
               state = 1;
            }
            else if(input[i] == 'b'){
               state = 0;
            }
            else{
               state = 3;
```

```
    }
    break;

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

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

case 3:
    state = 3;
    break;

default:
```

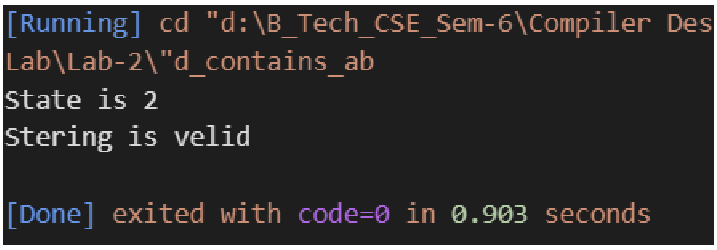
```
                break;
            }

            i++;
        }

        printf("State is %d\n",state);

        if(state == 2 ){
            printf("Stering is velid\n");
        }
        else{
            printf("Stering is Invelid\n");
        }



        return 0;
    }
```

INPUT: -



```
aaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbabababbbaabaabab
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Des
Lab\Lab-2\"d_contains_ab
State is 2
Stering is velid

[Done] exited with code=0 in 0.903 seconds
```

PRACTICAL: - 2

AIM:

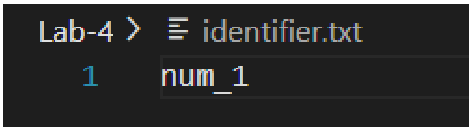
a). Write a program to recognize the valid identifiers.

PROGRAM CODE: -

```
#include <stdio.h>

#include <ctype.h>

#include <string.h>


// List of C keywords

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] == '_'))
```

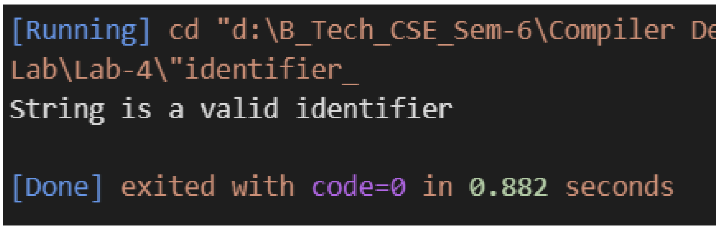
```
        return 0;

    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: -



```
1    num_1
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler De
Lab\Lab-4\"identifier_
String is a valid identifier

[Done] exited with code=0 in 0.882 seconds
```

AIM:
b). 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;
```

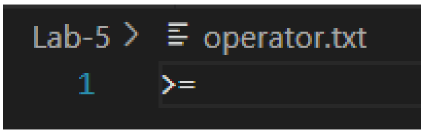
```
}
else if(input[i] == '/'){
    state = 12;
}
else if(input[i] == '%'){
    state = 15;
}
else if(input[i] == '&'){
    state = 18;
}
else if(input[i] == '|'){
    state = 21;
}
else if(input[i] == '<'){
    state = 24;
}
else if(input[i] == '>'){
    state = 28;
}
else if(input[i] == '!'){
    state = 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");}
```

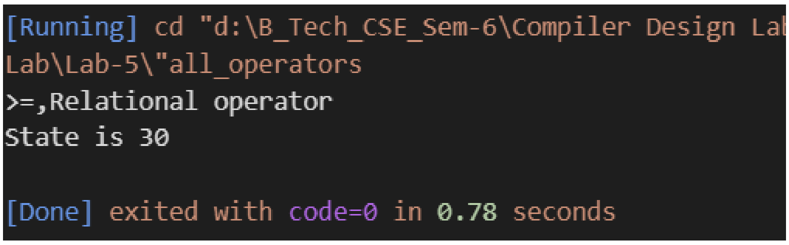
```
    else if(state == 15){printf("%,arithmetic operator\n");}
    else if(state == 18){printf("&,Bitwise operator\n");}
    else if(state == 21){printf("|,Bitwise operator\n");}
    else if(state == 24){printf("<,Relational operator\n");}
    else if(state == 28){printf(">,Relational operator\n");}
    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: -



```
>=
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Design La
Lab\Lab-5\"all_operators
>=,Relational operator
State is 30

[Done] exited with code=0 in 0.78 seconds
```

AIM:

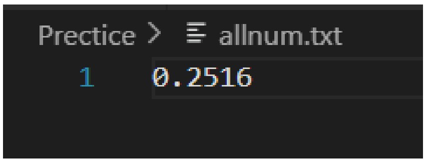
c). Write a program to recognize the valid number.

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("allnum.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;
```

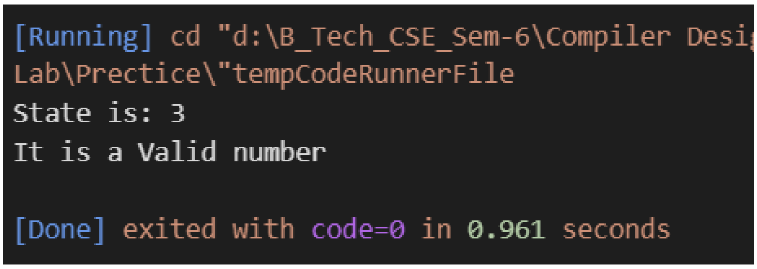
```
                    hasExponent = 1;
                } else {
                    state = 5;
                }
                break;

            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: -



```
0.2516
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Desi
Lab\Prectice\"tempCodeRunnerFile
State is: 3
It is a Valid number

[Done] exited with code=0 in 0.961 seconds
```

AIM:
d). Write a program to recognize the valid comments.

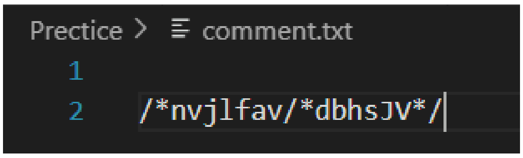
PROGRAM CODE: -

```
#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 Velid Comment\n");
    }
    else{
        printf("It is not Velid Comment\n");
        return 0;
    }
}
```

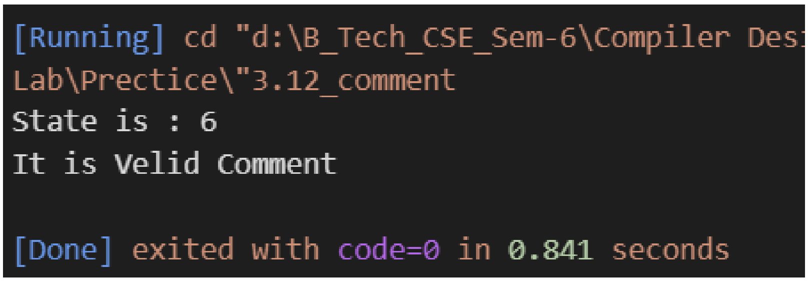
INPUT: -



Precticе > comment.txt

```

/*nvjlfav/*dbhsJV*/
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Des
Lab\Prectice\"3.12_comment
State is : 6
It is Velid Comment

[Done] exited with code=0 in 0.841 seconds
```

AIM:

e). 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') {
```

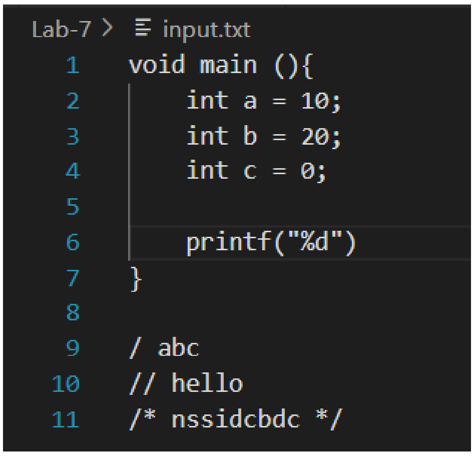
```
switch (state) {
  case 0:
    c = buffer[f];
    if (isalpha(c) || c == '_') {
      state = 1;
      lexeme[i++] = c;
    } else if (isdigit(c)) {
      state = 2;
      lexeme[i++] = c;
    } else if (c == '/') {
      state = 3;
    } else if (c == ' ' || c == '\t' || c == '\n') {
      state = 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++;
    }
}
```

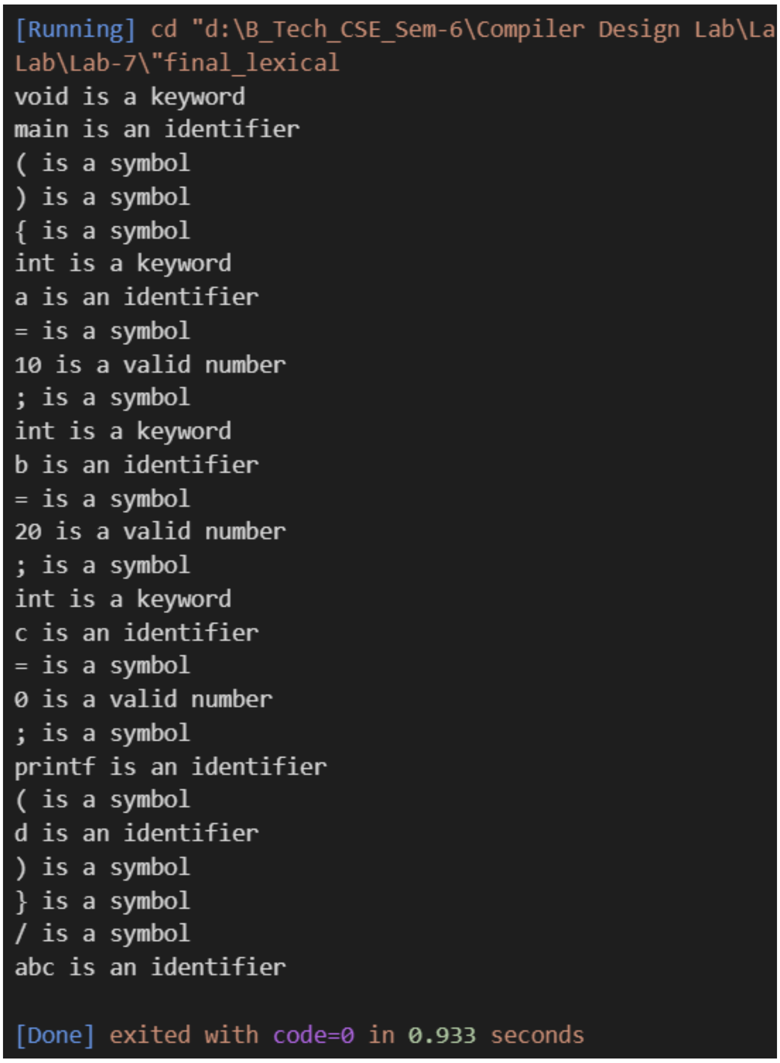
```
void check(char *lexeme) {
    char *keywords[] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do",
        "double", "else", "enum", "extern", "float", "for", "goto", "if",
        "inline", "int", "long", "register", "restrict", "return", "short", "signed",
        "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: -



```
void main (){
    int a = 10;
    int b = 20;
    int c = 0;

    printf("%d")
}

/ abc
// hello
/* nssidcbdc */
```

OUTPUT: -



```
[Running] cd "d:\B_Tech_CSE_Sem-6\Compiler Design Lab\La
Lab\Lab-7\"final_lexical
void is a keyword
main is an identifier
( is a symbol
) is a symbol
{ is a symbol
int is a keyword
a is an identifier
= is a symbol
10 is a valid number
; is a symbol
int is a keyword
b is an identifier
= is a symbol
20 is a valid number
; is a symbol
int is a keyword
c is an identifier
= is a symbol
0 is a valid number
; is a symbol
printf is an identifier
( is a symbol
d is an identifier
) is a symbol
} is a symbol
/ is a symbol
abc is an identifier

[Done] exited with code=0 in 0.933 seconds
```

PRACTICAL: - 3

AIM: To Study about Lexical Analyzer Generator (LEX) and Flex (Fast Lexical Analyzer

What is a Lexical Analyzer?

A lexical analyzer (or lexer) is the first phase of a compiler. It reads source code and splits it into tokens — such as keywords, identifiers, literals, and operators — for the parser.

What is LEX?

LEX (Lexical Analyzer Generator):

- Developed by AT&T Bell Labs.
- A tool for generating lexical analyzers (scanners).
- Input: .l file (lex specification).
- Output: A C program (lex.yy.c) that performs lexical analysis.

Structure of a Lex Program:

```
%{
  // C declarations
%}


%%
  // Pattern    Action
  [0-9]+      { printf("Number: %s\n", yytext); }
  [a-zA-Z]+    { printf("Word: %s\n", yytext); }
  "+"         { printf("Plus Sign\n"); }
%%


int main() {
  yylex();
}
```

What is Flex?

Flex (Fast Lexical Analyzer):

- An enhanced, faster, open-source version of LEX.
- Compatible with LEX syntax but with more features and better performance.

Flex Output Flow:

1. Write lex.l file (lex program).
2. Run: flex filename.l → generates lex.yy.c.
3. Compile: gcc lex.yy.c → produces executable. (a.exe)
4. Run the executable: a.exe

Key Concepts

Concept	Description
yytext	Holds the current token matched by Flex.
yylex()	The function Flex calls repeatedly to match tokens.
Regular Expressions	Used to define token patterns ([0-9]+, [a-zA-Z_])
yyin	File pointer; can be used to change input from stdin to a file.

Example Use Case: Identifier Detection

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

%%
[a-zA-Z_][a-zA-Z0-9_]*    printf("Valid Identifier: %s\n", yytext);
[ \t\n]                ; // ignore whitespace
.                      printf("Invalid character: %s\n", yytext);
```

```
%%

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

PRACTICAL: - 4

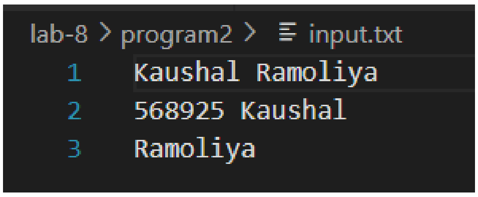
AIM:

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

PROGRAM CODE: -

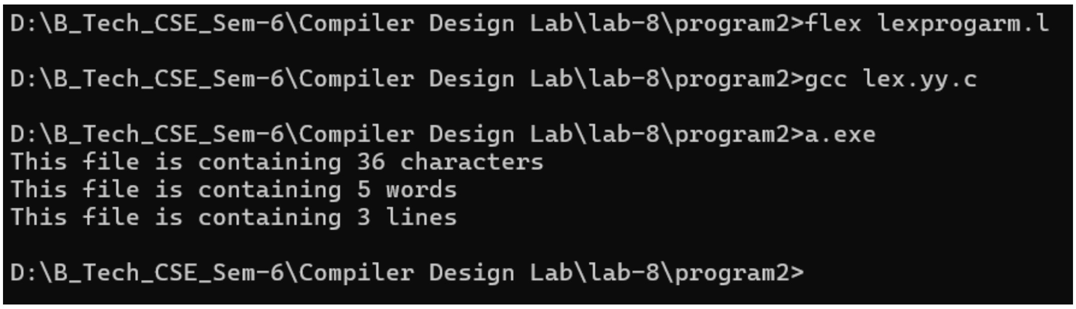
```
%{
#include<stdio.h>
int characters=0;
int words=0;
int lines=1;
%}
%%
[a-zA-z]    {characters++;}
" " {words++;}
\n  {lines++;words++;}
.   {characters++;}
%%
void main(){
yyin=fopen("input.txt","r");
yylex();
printf("This file is containing %d characters\n",characters);
printf("This file is containing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

INPUT: -



```
Kaushal Ramoliya
568925 Kaushal
Ramoliya
```

OUTPUT: -



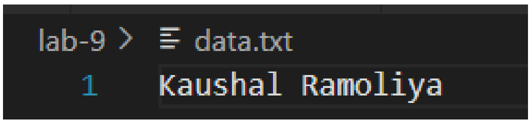
```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-8\program2>flex lexprogarm.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-8\program2>gcc lex.yy.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-8\program2>a.exe
This file is containing 36 characters
This file is containing 5 words
This file is containing 3 lines

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-8\program2>
```

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

PROGRAM CODE: -

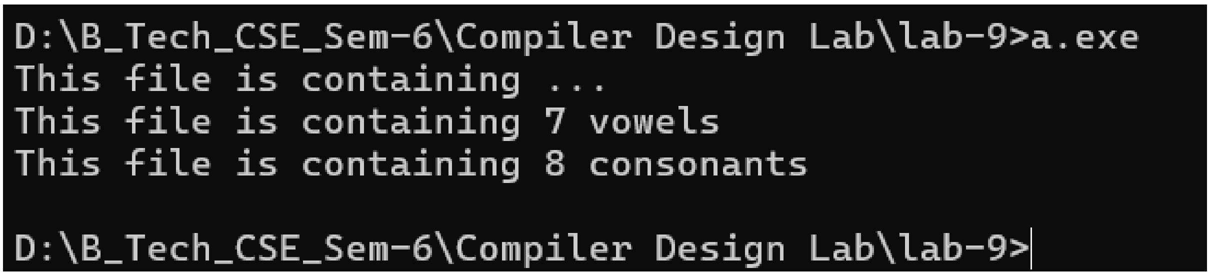
```
%{
#include<stdio.h>
int consonants=0, vowels = 0;
%}
%%
[aeiouAEIOU] {vowels++;}
[a-zA-Z]  {consonants++;}
\n        ;
.         ;
%%
void main(){
yyin=fopen("data.txt","r");
yylex();
printf("This file is containing ...\n");
printf("This file is containing %d vowels\n",vowels);
printf("This file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

INPUT: -



```
Kaushal Ramoliya
```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>a.exe
This file is containing ...
This file is containing 7 vowels
This file is containing 8 consonants

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>
```

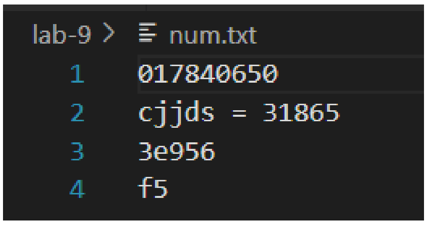
AIM:

c). Write a Lex program to print out all numbers from the given file.

PROGRAM CODE: -

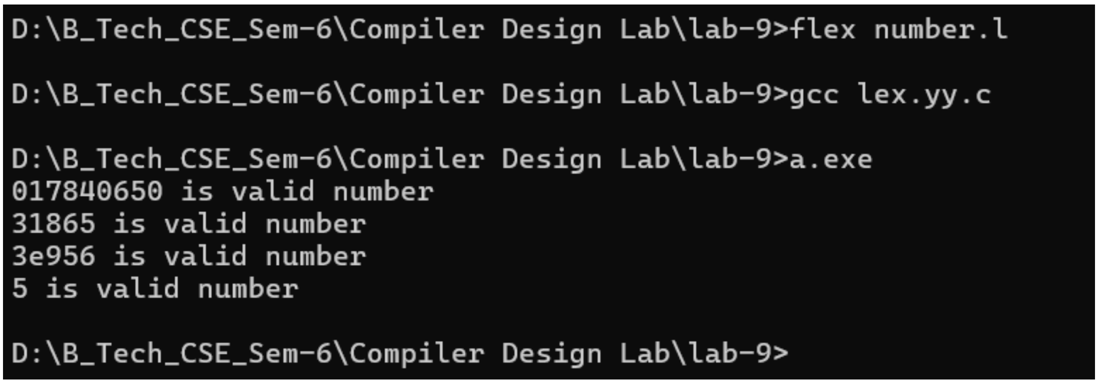
```
%{
#include<stdio.h>
%}
%%
[0-9]+(\.[0-9]+)?([Ee][+-]?[0-9]+)? printf("%s is valid number \n", yytext);
\n   ;
.   ;
%%
void main() {
   yyin = fopen("num.txt", "r");
   yylex();
   fclose(yyin);
}
int yywrap() { return 1; }
```

INPUT: -



```
017840650
cjjds = 31865
3e956
f5
```

OUTPUT: -



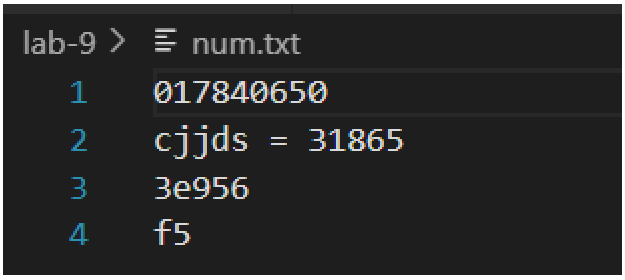
```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>flex number.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>gcc lex.yy.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>a.exe
017840650 is valid number
31865 is valid number
3e956 is valid number
5 is valid number

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>
```

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

PROGRAM CODE: -

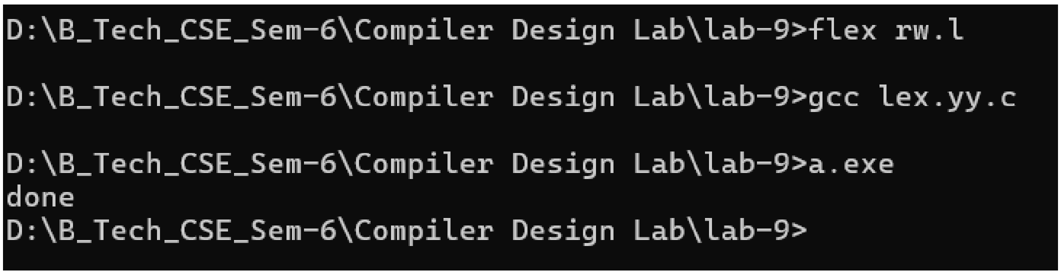
```
%{
#include<stdio.h>
int line_number=1;
%}
%%
.+ {fprintf(yyout,"%d: %s", line_number,yytext);line_number++;}
%%
int main(){
yyin=fopen("num.txt","r");
yyout=fopen("op.txt", "w");
yylex();
printf("done");
return 0;
}
int yywrap(){return(1);}
```

INPUT: -

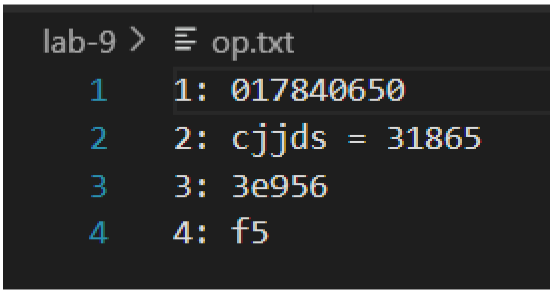


```
017840650
cjjds = 31865
3e956
f5
```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>flex rw.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>gcc lex.yy.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>a.exe
done
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\lab-9>
```



```
1: 017840650
2: cjjds = 31865
3: 3e956
4: f5
```

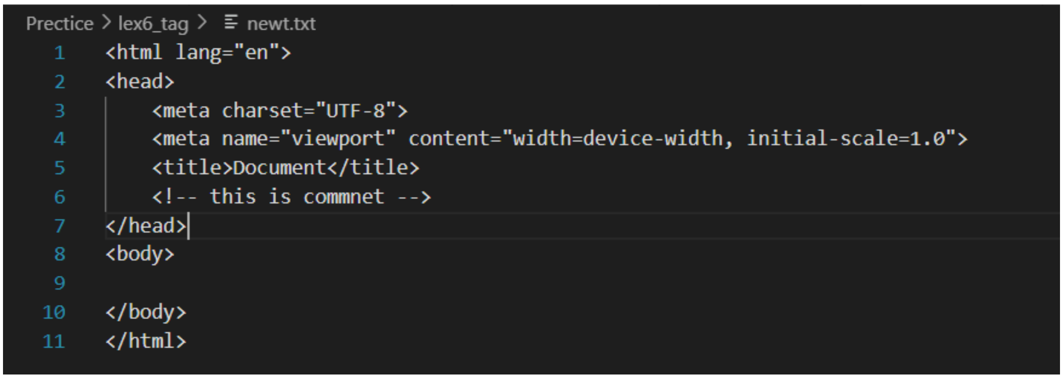
AIM:

e). 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("This is opening HTML tag : %s\n", yytext);}
"</"[A-Za-z0-9]+">" { printf("This is closing HTML tag : %s\n", yytext);}
"<!--"(.|\n)*"-->"  { printf("This is Comment HTML tag : %s\n", yytext);} {num++;}
.|\n|\t|[ ]  {  }
%%
void main(){
  yyin = fopen("newt.txt", "r");
  yylex();
  printf("%d\n", num);
  fclose(yyin);
}
int yywrap() {return(1);}
```

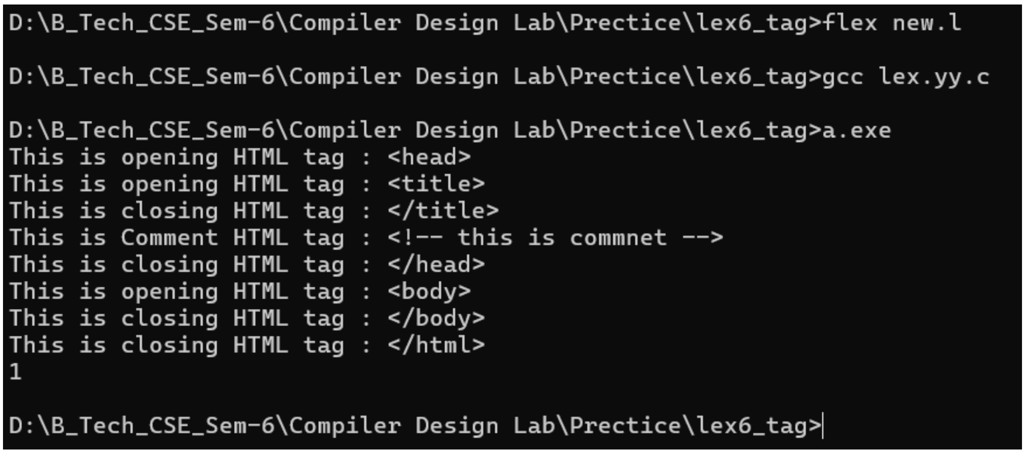
INPUT: -



Prectice > lex6_tag > newt.txt

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!-- this is commnet -->
</head>
<body>

</body>
</html>
```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex6_tag>flex new.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex6_tag>gcc lex.yy.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex6_tag>a.exe
This is opening HTML tag : <head>
This is opening HTML tag : <title>
This is closing HTML tag : </title>
This is Comment HTML tag : <!-- this is commnet -->
This is closing HTML tag : </head>
This is opening HTML tag : <body>
This is closing HTML tag : </body>
This is closing HTML tag : </html>
1

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex6_tag>
```

PRACTICAL: - 5

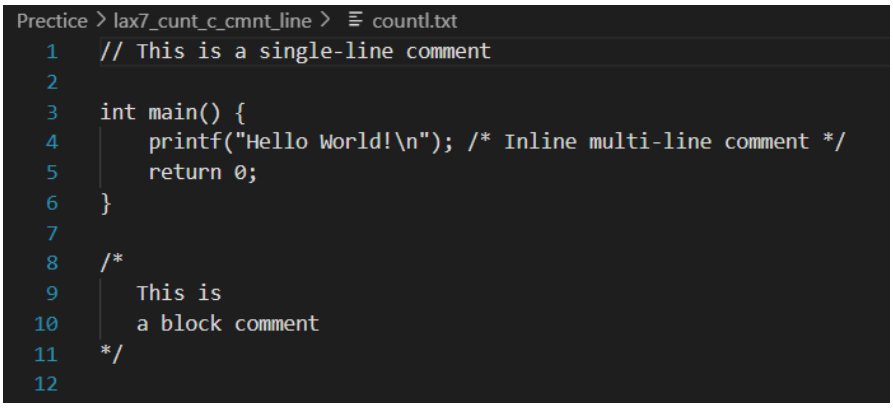
AIM:

a). 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: -

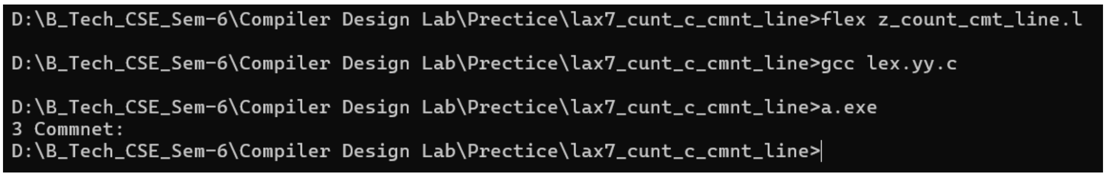
```
%{
#include<stdio.h>
int cmt = 0;
%}
%%
"//".*    { fprintf(yyout, "\n"); cmt++;}
"/*"([^*]|\*+[^/])*"*/"  { fprintf(yyout, "\n"); cmt++;}
.|\n   { fprintf(yyout, "%s", yytext);}
%%
void main(){
   yyin = fopen("countl.txt", "r");
   yyout = fopen("countlw.txt", "w");
   yylex();
   printf("%d Commnet: ", cmt);
}
int yywrap(){return(1);}
```

INPUT: -

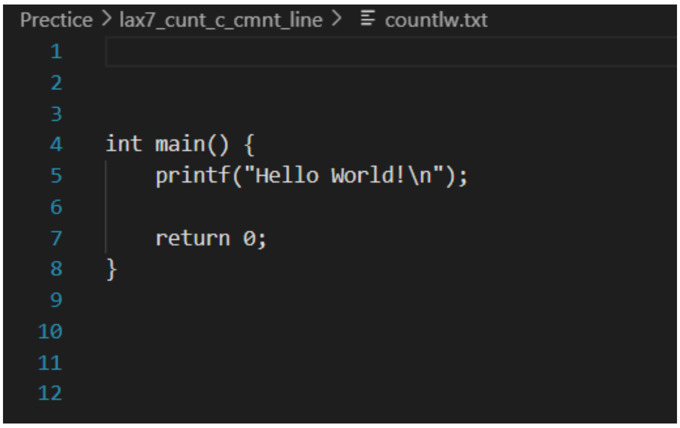


```
// This is a single-line comment

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

/*
   This is
   a block comment
*/

```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lax7_cunt_c_cmnt_line>flex z_count_cmt_line.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lax7_cunt_c_cmnt_line>gcc lex.yy.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lax7_cunt_c_cmnt_line>a.exe
3 Commnet:
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lax7_cunt_c_cmnt_line>
```



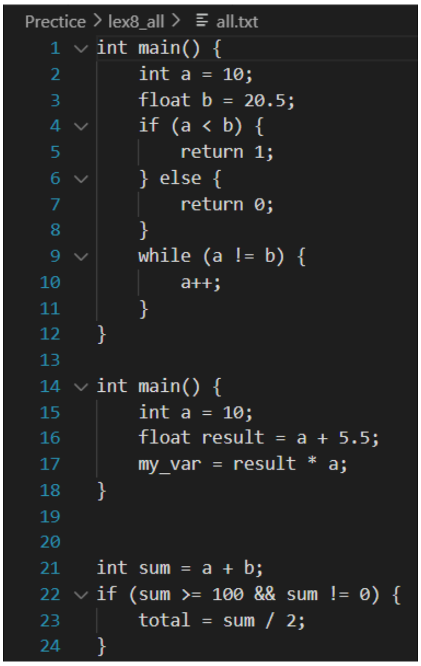
```



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

    return 0;
}
```

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

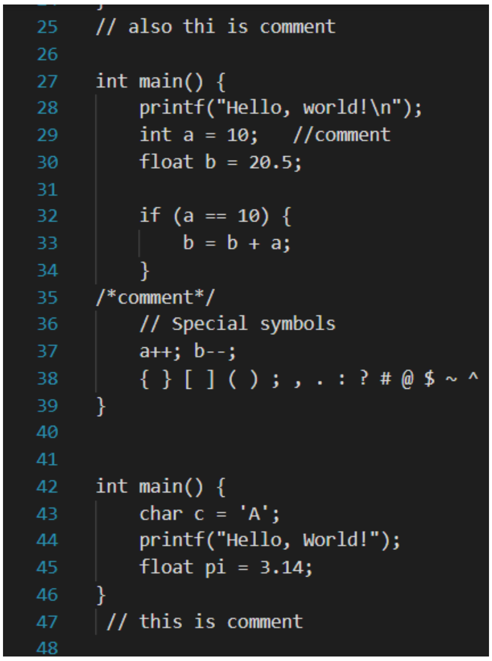
PROGRAM CODE: -

```
%{
  #include<stdio.h>
%}
%%
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|typ
edef|union|unsigned|void|volatile|while    { printf("<%s, Keyword>\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*  { printf("<%s, Identifier>\n", yytext);}
"=="|"!="|"<="|">="|"++"|"--"|"&&"|"||"  { printf("<%s, Operator>\n", yytext); }
[+\-*/%=<>&|!<>]                      { printf("<%s, Operator>\n", yytext); }
[0-9]+(\.[0-9]+)?([Ee][+-]?[0-9]+)?  { printf("<%s, Numbers>\n", yytext);}
[!@#$%^&*()~{}<>.;=]  { printf("<%s, Special symbol>\n", yytext);}
\"([^\\\"]|\\.)*\"   { printf("<%s, String Literal>\n", yytext); }
\'([^\\\']|\\.)\'    { printf("<%s, Char Literal>\n", yytext); }
.|\n|\t|[ ] { }
"//".*  {  }
"/*"([^*]\*+[^/])*"*/"   {   }
%%
int main(){
  yyin = fopen("all.txt", "r");
  yylex();
}
int yywrap(){return(1);}
```

INPUT: -

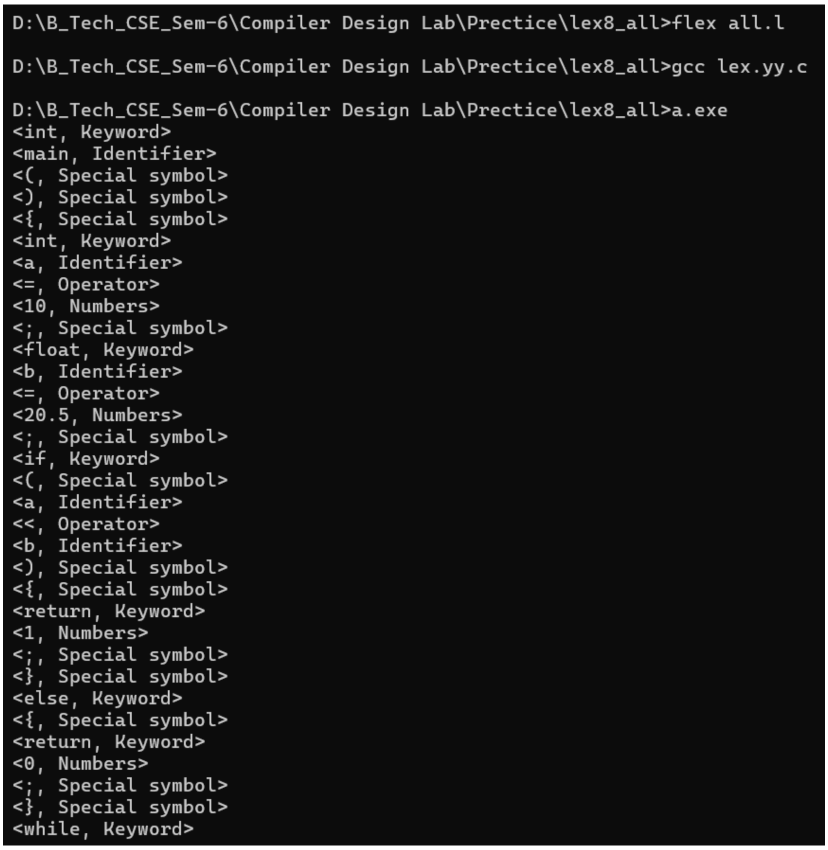


```
int main() {
    int a = 10;
    float b = 20.5;
    if (a < b) {
        return 1;
    } else {
        return 0;
    }
    while (a != b) {
        a++;
    }
}

int main() {
    int a = 10;
    float result = a + 5.5;
    my_var = result * a;
}


int sum = a + b;
if (sum >= 100 && sum != 0) {
    total = sum / 2;
}
```

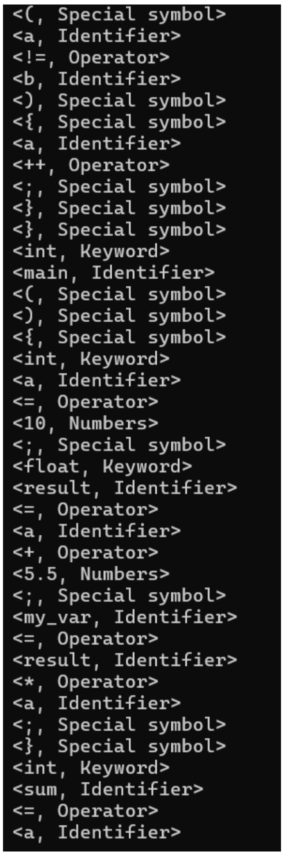


```
// also thi is comment

int main() {
    printf("Hello, world!\n");
    int a = 10;   //comment
    float b = 20.5;

    if (a == 10) {
        b = b + a;
    }
/*comment*/
    // Special symbols
    a++; b--;
    { } [ ] ( ) ; , . : ? # @ $ ~ ^
}


int main() {
    char c = 'A';
    printf("Hello, World!");
    float pi = 3.14;
}
 // this is comment
```

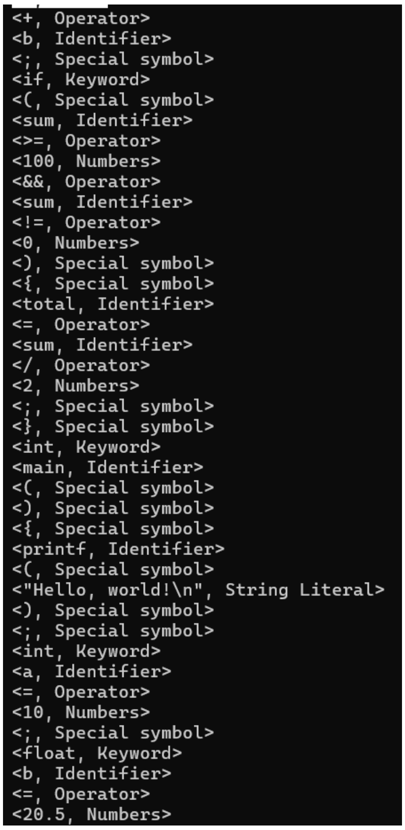
OUTPUT: -



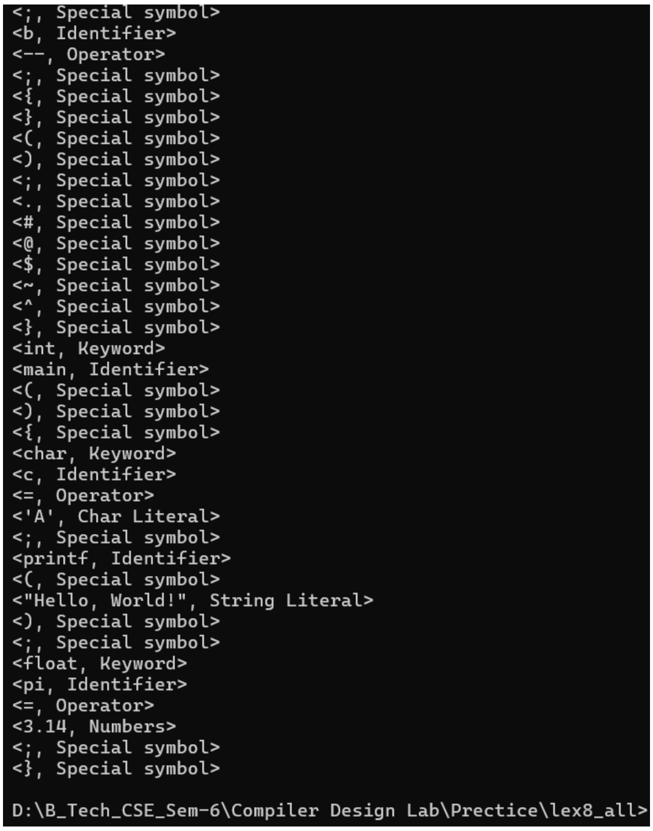
```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex8_all>flex all.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex8_all>gcc lex.yy.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex8_all>a.exe
<int, Keyword>
<main, Identifier>
<(, Special symbol>
<), Special symbol>
<{, Special symbol>
<int, Keyword>
<a, Identifier>
<=, Operator>
<10, Numbers>
<;, Special symbol>
<float, Keyword>
<b, Identifier>
<=, Operator>
<20.5, Numbers>
<;, Special symbol>
<if, Keyword>
<(, Special symbol>
<a, Identifier>
<<, Operator>
<b, Identifier>
<), Special symbol>
<{, Special symbol>
<return, Keyword>
<1, Numbers>
<;, Special symbol>
<}, Special symbol>
<else, Keyword>
<{, Special symbol>
<return, Keyword>
<0, Numbers>
<;, Special symbol>
<}, Special symbol>
<while, Keyword>
```



```
<(, Special symbol>
<a, Identifier>
<!=, Operator>
<b, Identifier>
<), Special symbol>
<{, Special symbol>
<a, Identifier>
<++, Operator>
<;, Special symbol>
<}, Special symbol>
<}, Special symbol>
<int, Keyword>
<main, Identifier>
<(, Special symbol>
<), Special symbol>
<{, Special symbol>
<int, Keyword>
<a, Identifier>
<=, Operator>
<10, Numbers>
<;, Special symbol>
<float, Keyword>
<result, Identifier>
<=, Operator>
<a, Identifier>
<+, Operator>
<5.5, Numbers>
<;, Special symbol>
<my_var, Identifier>
<=, Operator>
<result, Identifier>
<*, Operator>
<a, Identifier>
<;, Special symbol>
<}, Special symbol>
<int, Keyword>
<sum, Identifier>
<=, Operator>
<a, Identifier>
```



```
<+, Operator>
<b, Identifier>
<;, Special symbol>
<if, Keyword>
<(, Special symbol>
<sum, Identifier>
<>=, Operator>
<100, Numbers>
<&&, Operator>
<sum, Identifier>
<!=, Operator>
<0, Numbers>
<), Special symbol>
<{, Special symbol>
<total, Identifier>
<=, Operator>
<sum, Identifier>
</, Operator>
<2, Numbers>
<;, Special symbol>
<}, Special symbol>
<int, Keyword>
<main, Identifier>
<(, Special symbol>
<), Special symbol>
<{, Special symbol>
<printf, Identifier>
<(, Special symbol>
<"Hello, world!\n", String Literal>
<), Special symbol>
<;, Special symbol>
<int, Keyword>
<a, Identifier>
<=, Operator>
<10, Numbers>
<;, Special symbol>
<float, Keyword>
<b, Identifier>
<=, Operator>
<20.5, Numbers>
```



```
<;, Special symbol>
<b, Identifier>
<--, Operator>
<;, Special symbol>
<{, Special symbol>
<}, Special symbol>
<(, Special symbol>
<), Special symbol>
<;, Special symbol>
<., Special symbol>
<#, Special symbol>
<@, Special symbol>
<$, Special symbol>
<~, Special symbol>
<^, Special symbol>
<}, Special symbol>
<int, Keyword>
<main, Identifier>
<(, Special symbol>
<), Special symbol>
<{, Special symbol>
<char, Keyword>
<c, Identifier>
<=, Operator>
<'A', Char Literal>
<;, Special symbol>
<printf, Identifier>
<(, Special symbol>
<"Hello, World!", String Literal>
<), Special symbol>
<;, Special symbol>
<float, Keyword>
<pi, Identifier>
<=, Operator>
<3.14, Numbers>
<;, Special symbol>
<}, Special symbol>

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\Prectice\lex8_all>
```

PRACTICAL: - 6

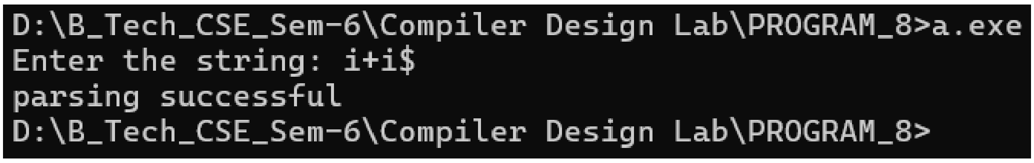
AIM: Program to implement Recursive Descent Parsing in C.

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 string");
    exit(1);
  }
  return 0;
}

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;
}
void main()
{
    printf("Enter the string: ");
    scanf("%s", &s);
    l = s[0];
    E();
    if (l == '$')
    {
        printf("parsing successful");
    }
    else
    {
        printf("Error while parsing the string\n");
    }
}
```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\PROGRAM_8>a.exe
Enter the string: i+i$
parsing successful
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\PROGRAM_8>
```

PRACTICAL: - 7

AIM:

a). To Study about Yet Another Compiler-Compiler (YACC).

What is YACC?

YACC stands for Yet Another Compiler-Compiler. It's a parser generator used in Unix-based systems. It works hand-in-hand with LEX (or Flex) to create a full compiler front-end.

How YACC Works

1. LEX handles lexical analysis: breaks source code into tokens.
2. YACC handles syntax analysis: checks grammar rules and parses tokens into a syntax tree.

YACC Program Structure

```
%{
   // C declarations (headers, variables)
%}


%token ID NUM


%%
// Grammar Rules Section
E : E '+' T  { printf("Matched: E + T\n"); }
  | T;


T : T '*' F  { printf("Matched: T * F\n"); }
  | F;


F : '(' E ')'
  | ID
  | NUM;
```

```
%%

// C Code Section
int main() {
    yyparse(); // Start parsing
    return 0;
}

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

YACC with LEX Workflow

Step	Command
1. Write lex.l	Your LEX file
2. Write yacc.y	Your YACC file
3. Run: yacc -d yacc.y	Generates y.tab.c and y.tab.h
4. Run: flex lex.l	Generates lex.yy.c
5. Compile: gcc y.tab.c lex.yy.c -o parser -lfl	
6. Run: ./parser	

Key YACC Components

Element	Description
%token	Declares tokens from LEX
yyparse()	Main parsing function
yyerror()	Called on syntax errors

Element	Description
yylval	Used to pass values from LEX to YACC
$$, $1, $2	Refer to values in grammar actions

Why Use YACC?

Automates parser creation
Works well with LEX/Flex
Helps build interpreters and compilers
Simplifies complex grammars

AIM:

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

PROGRAM CODE: -

Lex.l: -

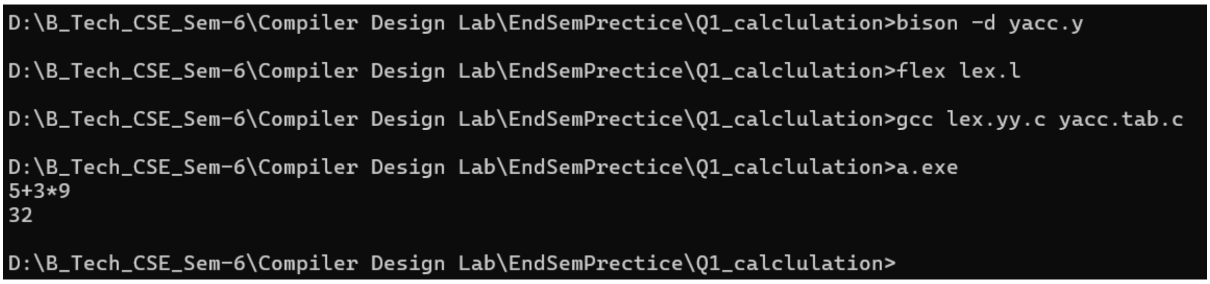
```
%{
#include <stdlib.h>
void yyerror(char *);
#include "yacc.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUM;}
[-+*\n] {return *yytext;}
[ \t] { }
. yyerror("invalid character");
%%
int yywrap() {
 return 0;
}
```

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: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q1_calclulation>bison -d yacc.y

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q1_calclulation>flex lex.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q1_calclulation>gcc lex.yy.c yacc.tab.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q1_calclulation>a.exe
5+3*9
32

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q1_calclulation>
```

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

PROGRAM CODE: -

Lex.l:-

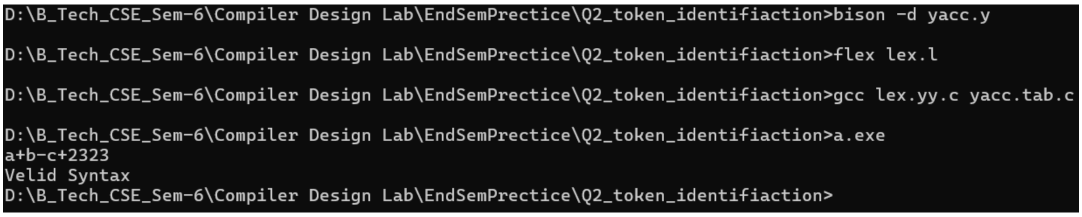
```
%{
 #include <stdlib.h>
 void yyerror(char *);
 #include "yacc.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext); return NUM; }
[a-zA-Z_][a-zA-Z_0-9]* {return ID; }
[-+*\n] {return *yytext; }
[ \t] {  }
. yyerror("Invelid Character");
%%
int yywrap(){
  return 0;
}
```

Yacc.y: -

```
%{
 #include <stdio.h>
 int yylex(void);
 void yyerror(char *);
%}
%token NUM
%token ID
%%
S : E '\n' { printf("Velid Syntax"); 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;
}
```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q2_token_identifiaction>bison -d yacc.y

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q2_token_identifiaction>flex lex.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q2_token_identifiaction>gcc lex.yy.c yacc.tab.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q2_token_identifiaction>a.exe
a+b-c+2323
Velid Syntax
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q2_token_identifiaction>
```

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

PROGRAM CODE: -

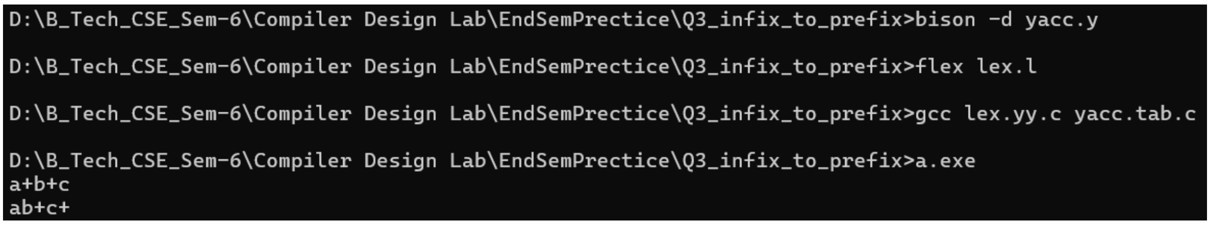
Lax.l: -

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

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;
}
```

OUTPUT: -



```
D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q3_infix_to_prefix>bison -d yacc.y

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q3_infix_to_prefix>flex lex.l

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q3_infix_to_prefix>gcc lex.yy.c yacc.tab.c

D:\B_Tech_CSE_Sem-6\Compiler Design Lab\EndSemPrectice\Q3_infix_to_prefix>a.exe
a+b+c
ab+c+
```