Lab Manual

OF

Compiler Design

Bachelor of Technology(CSE)

By

Harshil Brahmani

(22000744) 3rd Year,

Semester 5

Department of Computer Science and Engineering

School Engineering and Technology

Navrachana University, Vadodara

Autumn Semester (2022-2023)

1. **Write a program to recognize the string starting from ‘a’ over {a,b}.**

Code:

```
#include<stdio.h>
int main()
{
char string[100]; printf("Enter the string: "); scanf("%s", string);
int i=0, state=0; while(string[i]!='\0')
{
switch(state)
{
case 0: if(string[i]=='a') state=1; else if(string[i]=='b')state=2; else state=3;
break;
case 1: if(string[i]=='a') state=1;
else if(string[i]=='b') state=1;
else
state=3; break;
case 2: if(string[i]=='a') state=2;
else if(string[i]=='b') state=2;
else
state=3; break;
case 3: break;
} i+
+;
}
if(state==1)
printf("String accepted\n"); else if(state==2)
printf("String not accepted\n"); else
printf("String not recognized\n");

return 0;
}
```

2. **Write a program to recognize the string ending on ‘a’ over {a,b}.**

Code:

```
#include<stdio.h>
int main()
{
char string[100]; int state=0, i=0;
printf("Enter a string: "); scanf("%s", string);

while (string[i]!='\0')
{
switch(state)
{
case 0:
if(string[i]=='a') state=1;
else if(string[i]=='b') state=0; else state=2;
break; case 1:
if(string[i]=='a') state=1;
else if(string[i]=='b') state=0; else state=2;
break; case 2:
break;


} i++;
}
if(state==1)
printf("String accepted\n"); else if(state==0)
printf("String not accepted\n"); else
printf("String not recognized\n");

return 0;
}
```

3. **Write a program to recognize strings end with ‘ab’.**

   Take the input from text file.

Code:

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

int main() {
char string[100];
FILE *file = fopen("input.txt", "r"); if (file == NULL) {

printf("Could not open file input.txt\n"); return 1;
}
while (fgets(string, sizeof(string), file))
{
int i = 0, state = 0; while (string[i] != '\0')
{
switch (state) { case 0:
if (string[i] == 'a') state = 1;
else if (string[i] == 'b') state = 0; else state = 3;
break; case 1:
if (string[i] == 'a') state = 1;
else if (string[i] == 'b') state = 2; else state = 3;
break; case 2:
if (string[i] == 'a') state = 1;
else if (string[i] == 'b') state = 0; else state = 3;
break; case 3:

state = 3; break;
} i++;
}
if (state == 2)
printf("String accepted: %s\n", string); else if (state == 0 || state == 1)
printf("String not accepted: %s\n", string); else
printf("String not recognized: %s\n", string);
}
fclose(file); return 0;
```

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

<u>Code:</u>

```
#include<stdio.h>
int main()
{
char string[100];
FILE *file = fopen("4th.txt", "r"); if (file == NULL)
{  printf("Could not open file input.txt\n"); return
1;
}
while(fgets(string, sizeof(string), file))
{
int i=0, state=0; while(string[i]!='\0')
{
switch(state)
{
case 0:
if(string[i]=='a') state=1;
else if(string[i]=='b') state=0; break;
case 1:
if(string[i]=='a') state=1;
else if(string[i]=='b') state=2; break;
case 2:
if(string[i]=='a') state=1;
else if(string[i]=='b') state=2; break;
} i+
+;
}
if(state==2)
printf("String accepted: %s\n", string); else if(state==0 || state==1)
printf("String not accepted: %s\n", string); else
printf("String not recognized: %s\n", string);
}
```

5. Single line

   comment

Code:

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

int main()
{
char string[100];
FILE *file = fopen("comment.txt", "r"); if (file == NULL)
{  printf("Could not open file input.txt\n"); return 1;
}
while (fgets(string, sizeof(string), file))
{
int i=0, state=0; while(string[i]!='\0')
{
switch(state)
{
case 0:
if(string[i]=='/') state=1; else state=2;
break; case 1:
if(string[i]=='/') state=3; else state=2;
break; case 2:

break; case 3:
break;
} i+
+;
}
if(state==3)
printf("Comment valid: %s\n", string); else
printf("Comment not valid: %s\n", string);
}


}
```

6. MultilineComment

Code:

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

int main()
{
char string[1000];
FILE *file = fopen("comment2.txt", "r"); if (file == NULL)
{  printf("Could not open file input.txt\n"); return 1;
}
while (fgets(string, sizeof(string), file))
{
int i=0, state=0; while(string[i]!='\0')
{
switch(state)
{
case 0:
if(string[i]=='/') state=1;

else state=2; break;
case 1:
if(string[i]=='/') state=3;
else if (string[i]=='*') state=4; else state=2;
break; case 2:
break; case 3:
break; case 4:
if (string[i]=='*') state=5; else state=4;
case 5:
if(string[i]=='/') state=6; else state=4;
case 6:
break;
}  i+
+;
}
if(state==3)
printf("Singleline Comment valid: %s\n", string); else if(state==6)
printf("Multiline Comment valid: %s\n", string); else
printf("Comment not valid: %s\n", string);
}
```

a. Write a program to recognize the valid identifiers.

<u>Code:</u>

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<stdbool.h>

int main()
{
char string[1000]; printf("Enter the string: ");
scanf("%s", string); // Prevents buffer overflow int i = 0, state = 0;
int num=1; while (num>0)
{
switch (state)
{
case 0: //A
if (string[i] == 'i') state = 1;
else if (isalpha(string[i]) || string[i] == '_') state = 5; break;

case 1: //B
if (string[i] == 'n') state = 2;
else if (isalpha(string[i]) || string[i] == '_' || isdigit(string[i])) state = 5;
else state = 6; break;

case 2: //C
if (string[i] == 't') state = 3;
else if (isalpha(string[i]) || string[i] == '_' || isdigit(string[i])) state = 5;
else state = 6; break;

case 3: //D
if (string[i] == '\0') state = 4;
else if (isalpha(string[i]) || isdigit(string[i]) || string[i] ==

'_') state = 5;



num--; break;
```

b. **Write a program to recognize the valid operators.**

Code:

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

int main()
{
char string[1000]; printf("Enter the string: ");
scanf("%s", string); // Prevents buffer overflow int i = 0, state = 0;
int num=1; while (num>0)
{
switch(state)
{
case 0:
if (string[i] == '+') state = 51; else if(string[i] == '*') state = 51; else if(string[i] == '/') state = 60; else if(string[i] == '=')
state = 53; else if(string[i] == '-') state = 54; else if(string[i] == '?') state = 55; else if(string[i] == '<') state = 56; else
if(string[i] == '>') state = 59; else if(string[i] == '!') state = 57; else if(string[i] == '&') state = 58; else if(string[i] == '|')
state = 61;
else if(string[i] == '~' || string[i] == '^') state = 62; break;

case 51: // Arithmetic
if (string[i] == '\0') state = 51; else if (string[i] == '+') state = 52; else if (string[i] == '=') state = 53; else state = 0;
num--; break;

case 52: // Unary break;

case 53: // Assignment

if(string[i] == '=') state = 56; else state = 0;
num--; break;

case 54: // -
if (string[i] == '\0') state = 51; else if (string[i] == '-') state = 52; else if (string[i] == '=') state = 53; else state = 0;
num--; break;

case 55: // Ternary
if (string[i] == ':') state = 55; else if(string[i] == '\0') state = 0; else state = 0;
num--; break;

case 56: //Relational
if (string[i] == '=') state = 56; else if(string[i] == '<') state = 58; else state = 0;
num--; break;

case 59: // >
if (string[i] == '>') state = 58; else if (string[i] == '=') state = 56; else state = 0;
num--; break;

case 57: // Logical
if (string[i] == '=') state = 56; else if(string[i] == '\0') state = 57; else state = 0;
num--; break;

case 58: // Bitwise
if (string[i] == '&') state = 57; else if(string[i] == '\0') state = 58; else state = 0;

num--; break;

case 60: // "/"
if (string[i] == '\0') state = 51; else state = 0;
num--; break;

case 61: // "|"
if (string[i] == '|') state = 57; else if(string[i] == '\0') state = 58; else state = 0;
num--; break;

case 62: // ~ ^
if (string[i] == '\0') state = 58; else state = 0;
num--; break;
} i+
+;
}

if (state == 51) printf("%s is an Arithmatic operator.", string); else if(state == 52) printf("%s is an Unary operator.",
string); else if(state == 53) printf("%s is an Assignment operator.", string); else if(state == 55) printf("%s is ternary or
conditional operator.",
string);
else if(state == 56) printf("%s is a Relational Operator.", string); else if(state == 57) printf("%s is a Logical operator.",
string); else if(state == 58) printf("%s is a Bitwise operator.", string); else
printf("Processing  ");

return 0;
```

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

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>

int main()
{  FILE *file;
char buffer[100];
char lexeme[100];
char c;
int f, i, state;

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

while (fgets(buffer, 100, file)) { buffer[strcspn(buffer, "\n")] = 0; f = 0; i
= 0;
state = 0;

while (buffer[f] != '\0')
switch (state) {
case 0:
c = buffer[f];
if (isdigit(c)) { state = 40; lexeme[i++] = c; }
else { state = 0; }
break;

case 40:
c = buffer[f];
if (isdigit(c)) { state = 40; lexeme[i++] = c; }
else if (c == '.') { state = 41; lexeme[i++] = c; }
else if (c == 'E' || c == 'e') { state = 43; lexeme[i++] = c;
}
else {
lexeme[i] = '\0';
printf("%s is a valid number\n", lexeme);

i = 0;
state = 0; f--;
}
break;
```

```
case 41:
c = buffer[f];
if (isdigit(c)) { state = 42; lexeme[i++] = c; } else
{  lexeme[i] = '\0';
printf("%s is an invalid number (expected digit after

decimal)\n", lexeme);




}



i = 0;
state = 0; f--;

break;

case 42:
c = buffer[f];
if (isdigit(c)) { state = 42; lexeme[i++] = c; }
else if (c == 'E' || c == 'e') { state = 43; lexeme[i++] = c;
}
else {
lexeme[i] = '\0';
printf("%s is a valid number\n", lexeme); i = 0;
state = 0; f--;
}
break;

case 43:
c = buffer[f];
if (c == '+' || c == '-') { state = 44; lexeme[i++] = c; } else if (isdigit(c)) { state = 45; lexeme[i++] = c; } else
{  lexeme[i] = '\0';
printf("%s is an invalid number (expected digit or sign after 'E'/'e')\n", lexeme);
i = 0;
state = 0; f--;
}

break;

case 44:
c = buffer[f];
if (isdigit(c)) { state = 45; lexeme[i++] = c; } else
{  lexeme[i] = '\0';
printf("%s is an invalid number (expected digit after sign in exponent)\n", lexeme);
i = 0;
state = 0; f--;
}
```

```
break;

case 45:
c = buffer[f];
if (isdigit(c)) { state = 45; lexeme[i++] = c; } else
{  lexeme[i] = '\0';
printf("%s is a valid number\n", lexeme); i = 0;
state = 0; f--;



} f++;
}

}
break;


if (state == 40 || state == 41 || state == 42 || state == 45) { lexeme[i] = '\0';
printf("%s is a valid number\n", lexeme);
} else {
printf("%s is an invalid number\n", buffer);
}
}

fclose(file); return
```

d. **Write a program to recognize the valid comments.**

Code:

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

int main()
{
    char string[1000];
    FILE *file = fopen("comment2.txt",
    "r"); if (file == NULL) {
        printf("Could not open file
        input.txt\n"); return 1;
    }
    while (fgets(string, sizeof(string), file))
    {
        int i=0, state=0; while(string[i]!
        ='\0')
        {
            switch(state)
            {
                case 0:
                    if(string[i]=='/') state=1;
                    else state=2;
                    break
                ; case 1:
                    if(string[i]=='/') state=3;
                    else if (string[i]=='*') state=4;
                    else state=2;
                    break
                case 4:
                    if (string[i]=='*') state=5;
                    else state=4;
                case 5:
                    if(string[i]=='/') state=6;
                    else state=4;
                case 6:
                    break;
            }
            i+
            +;
        }
        if(state==3)
            printf("Singleline Comment valid: %s\n",
        string); else if(state==6)
            printf("Multiline Comment valid: %s\n", string);
        else
            printf("Comment not valid: %s\n", string);
    }
return
```

e. **Program to implement**

Lexical Analyzer.

f. Code:

```
#include <stdio.h> #include
<stdlib.h> #include <ctype.h>
#include <string.h> #define
BUFFER_SIZE 1000 void
check(char *lexeme);

int main() {
        FILE *f1;
        char buffer[BUFFER_SIZE], lexeme[50]; // Static buffer for input and lexeme storage
        char c;
        int f = 0, state = 0, i = 0; f1 =
        fopen("Input.txt", "r");
        fread(buffer, sizeof(char), BUFFER_SIZE - 1, f1);
        buffer[BUFFER_SIZE - 1] = '\0'; // Null termination fclose(f1);

        while (buffer[f] != '\0') { c =
                buffer[f];
                switch (state) { case 0:
                                if (isalpha(c) || c == '_') { state = 1;
                                        lexeme[i++] = c;
                                }
                                else if (c == ' ' || c == '\t' || c == '\n') { state = 0;
                                }
                                else if(isdigit(c)) { state = 13;
                                        lexeme[i++] = c;
                                }
                                else if (c == '/') {
                                        state = 11;             // For comment
                                }
                                else if (c == ';' || c == ',' || c == '{' || c == '}') { printf(" %c is a symbol\n", c);
                                        state = 0;
                                }
                                else if (strchr("+-*/=%?<>!&|~^", c)) {
```

```
                        state  = 50; lexeme[i+
                        +] = c;
                }
                else {
                        state = 0;
                }
                break;

        case 1:
                if (isalpha(c) || isdigit(c) || c == '_') { state = 1; lexeme[i+
                        +] = c;
                } else {
                        lexeme[i] = '\0'; // Null-terminate the lexeme check(lexeme);   // Check if
                        it's a keyword or identifier state  = 0;
                        i  =  0;
                        f--; // Step back to reprocess the current non-alphanumeric


character
                }
                break;

        case 13:
                if(isdigit(c)) { state = 13; lexeme[i+
                        +] = c;

                }
                else if(c=='.') { state=14; lexeme[i+
                        +]=c;
                }
                else if(c=='E'||c=='e') { state=16;
                        lexeme[i++]=c;
                }
                else {
                        lexeme[i]='\0';
                        printf("%s is a valid number\n", lexeme); i=0;
                        state=0; f-
                        -;
                }
                break;

        case 50: // Operator Handling
```

```
switch (lexeme[0]) { case
    '+':
        if (c == '+') {
            printf("%s is a Unary operator\n", lexeme); state = 0;
        }
        else if (c == '=') {
            printf("%s is an Assignment operator\n", lexeme); state = 0;
        }
        else {
            printf("%s is an Arithmetic operator\n", lexeme); state = 0;
            f--;
        }
        break;

    case '-':
        if (c == '-') {
            printf("%s is a Unary operator\n", lexeme); state = 0;
        }
        else if (c == '=') {
            printf("%s is an Assignment operator\n", lexeme); state = 0;
        }
        else {
            printf("%s is an Arithmetic operator\n", lexeme); state = 0;
            f--;
        }
        break;

    case '*':
    case '/':
    case '%':
        if (c == '=') {
            printf("%s is an Assignment operator\n", lexeme); state = 0;
        }
        else {
            printf("%s is an Arithmetic operator\n", lexeme); state = 0;
            f--;
        }
        break;

    case '=':
        if (c == '=') {
            printf("%s is a Relational operator\n", lexeme); state = 0;
        }
        else {
            printf("%s is an Assignment operator\n", lexeme); state = 0;
            f--;
        }
        break;

    case '<':
    case '>':
        if (c == '=' || c == lexeme[0]) {
            printf("%s is a Relational operator\n", lexeme); state = 0;
        }
        else {
            printf("%s is a Relational operator\n", lexeme); state = 0;
            f--;
        }
        break;

    case '!':
    case '&':
    case '|':
        if (c == '=') {
            printf("%s is a Logical operator\n", lexeme); state = 0;
        }
        else if (c == lexeme[0]) {
            printf("%s is a Logical operator\n", lexeme); state = 0;
        }
        else {
            printf("%s is a Logical operator\n", lexeme); state = 0;
            f--;
        }
        break;
```

```
                                        case '~':
                                        case '^':
                                                printf("%s is a Bitwise operator\n", lexeme); state = 0;
                                                f--;
                                                break;


                                        case '?':
                                                if (c == ':') {
                                                        printf("%s is a Ternary or conditional operator\n",
lexeme);

                                                        state = 0;

                                                }
                                                else {
                                                        state = 0; f--;
                                                }
                                                break;

                                        default:
                                                state = 0;
                                                break;
                                }
                                lexeme[0] = '\0'; i =
                                0;
                                break;

                        default:
                                state = 0; break;
                } f+
                +;
        }
}

void check(char *lexeme) { char
        *keywords[] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do",
        "double", "else", "ef", "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);
```

4.Implement following programs using Lex.

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

Code:

```
%{

#include<stdio.h>

int

words=0,characters

=0,n o_of_lines=0;

%}

%%

\n

{no_of_lines++,words++;

}

. characters++; [\t ]+

words++;

%%

void main(){

yyin =

fopen("4_1.txt","

r"); yylex();
printf("This file is
containing %d
words.\n",words);
printf("This file is
containing %d
characters.
\n",character
s); printf("This file
is containing %d
no_of_lines.
\n",no_of_lin es);

}

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

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

Code:

```
%{

#include<stdio.h>

int vowels=0, consonant=0;

%}

%%

[aeiouAEIOU]

vowels++; [a-zA-Z]

consonant++;

.  ;

\n  ;

%%

void main(){

yyin =

fopen("input.txt","r");

yylex();

printf("This file is containing %d vowels.

\n",vowels); printf("This file is containing %d

consonants.\n",consonant);

}

int yywrap()

{ return(1);}
```

c.**Write a Lex program to print out all numbers**

from the given file.

Code:

```
%{
 #include<stdio.h>
%}
%%
[0-9]+(.[0-9]+)?([eE][+-]?[0-9]+)?  printf("%s is valid number \n",yytext);
\n      ;
.       ;
%%

void main() {
yyin =
fopen("input.txt","r");
yylex();
}
int yywrap()
{return(1);}
```

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

Code:

```
%{
int line_number = 1;
%}
%%
.+ {fprintf(yyout,"%d: %s",line_number,yytext);line_number++;}
%%
int main() {
yyin = fopen("input.txt","r");
yyout = fopen("op.txt","w");
yylex();
printf("Done");
return 0;
}
int yywrap()

{return(1);}
```

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

Code:

```
%{
#include<stdio.h
> int num=0;
%}
%%
"<"[A-Za-z0-9]+">"|"<"[/A-Za-z0-9]+">" printf("%s is valid markup tag
\n",yytext); "<!--"[A-Za-z ]*"-->" num++;
.|\n ;
%%
int main() {
yyin =
fopen("htmlfile.txt","r");
yylex();
printf("%d
comment",num);
return 0;
}
int yywrap(){return(1);}
```

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.**

<u>Code:</u>

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

int comment_line_count = 0;
FILE *output_file;
int in_comment_line = 0;
%}

%x SINGLE_LINE_COMMENT
%x MULTI_LINE_COMMENT

%%
"//"              { BEGIN(SINGLE_LINE_COMMENT); in_comment_line = 1; }
<SINGLE_LINE_COMMENT>\n {
                  if (in_comment_line) {
                     comment_line_count++;
                     in_comment_line = 0;
                  }
                  fprintf(output_file, "\n");
                  BEGIN(INITIAL);
                 }
<SINGLE_LINE_COMMENT>.  { /* Ignore comment content */ }

"/*"              { BEGIN(MULTI_LINE_COMMENT); in_comment_line = 1; }
<MULTI_LINE_COMMENT>\n  {
                  if (in_comment_line) {
                     comment_line_count++;
                     in_comment_line = 0;
                  }
                  fprintf(output_file, "\n");
                  in_comment_line = 1;
                 }
<MULTI_LINE_COMMENT>"*/" { BEGIN(INITIAL); in_comment_line = 0; }
<MULTI_LINE_COMMENT>.   { /* Ignore comment content */ }

\n                { fprintf(output_file, "\n"); }
.               { fprintf(output_file, "%s", yytext); }
%%

int main(int argc, char *argv[]) {
    FILE *input_file;

    if (argc != 3) {
        printf("Usage: %s input_file output_file\n", argv[0]);
        return 1;
    }

    input_file = fopen(argv[1], "r");
    if (!input_file) {
        printf("Could not open input file %s\n", argv[1]);
        return 1;
    }

    output_file = fopen(argv[2], "w");
    if (!output_file) {
        printf("Could not create output file %s\n", argv[2]);
        fclose(input_file);
        return 1;
    }

    yyin = input_file;
    yylex();

    printf("Total comment lines found: %d\n", comment_line_count);
    printf("Cleaned code has been written to %s\n", argv[2]);

    fclose(input_file);
    fclose(output_file);
    return 0;
}

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

SAMPLE C :-

```
// Sample C program with comments
#include <stdio.h>

// This function prints "Hello, World!"
void hello() {
    printf("Hello, World!\n");  // Print greeting
}

/*
 * This is a multi-line comment
 * that spans over multiple
 * lines of code
 */
int main() {
    // Call hello function
    hello();

    /* Another comment */
    return 0;
}
```

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

Code:

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

%x STRING_LITERAL
%x CHAR_LITERAL
%x SINGLE_LINE_COMMENT
%x MULTI_LINE_COMMENT

%%
[ \t\n]+            { /* ignore whitespace */ }

"//"                { BEGIN(SINGLE_LINE_COMMENT); }
<SINGLE_LINE_COMMENT>\n    { BEGIN(INITIAL); }
<SINGLE_LINE_COMMENT>.    { /* ignore comment content */ }

"/*"                { BEGIN(MULTI_LINE_COMMENT); }
<MULTI_LINE_COMMENT>"*/"   { BEGIN(INITIAL); }
```

```
<MULTI_LINE_COMMENT>.|\n    { /* ignore comment content */ }

\"                    { printf("STRING_START\n"); BEGIN(STRING_LITERAL); }
<STRING_LITERAL>\\\"      { /* escaped quote in string */ }
<STRING_LITERAL>\"        { printf("STRING_END\n"); BEGIN(INITIAL); }
<STRING_LITERAL>.|\n      { /* string content */ }

\'                    { printf("CHAR_START\n"); BEGIN(CHAR_LITERAL); }
<CHAR_LITERAL>\\\'        { /* escaped quote in char */ }
<CHAR_LITERAL>\'          { printf("CHAR_END\n"); BEGIN(INITIAL); }
<CHAR_LITERAL>.           { /* char content */ }

"auto"|"break"|"case"|"char"|"const"|"continue"|"default"|"do"|"double"|"else"|"en
um"|"extern"|"float"|"for"|"goto"|"if"|"int"|"long"|"register"|"return"|"short"|"sign
ed"|"sizeof"|"static"|"struct"|"switch"|"typedef"|"union"|"unsigned"|"void"|"volatil
e"|"while"    { printf("KEYWORD: %s\n", yytext); }

[a-zA-Z_][a-zA-Z0-9_]*     { printf("IDENTIFIER: %s\n", yytext); }

[0-9]+                { printf("INTEGER: %s\n", yytext); }
[0-9]+\.[0-9]+([eE][+-]?[0-9]+)?  { printf("FLOAT: %s\n", yytext); }
0[xX][0-9a-fA-F]+         { printf("HEX_INTEGER: %s\n", yytext); }

"+"|"-"|"*"|"/"|"%"|"++"|"--"|"=="|"!
="|">"|"<"|">="|"<="|"&&"|"||"|"!"|"&"|"|"|"^"|"~"|"<<"|">>"|"="|"+="|"-="|"*="|"/
="|"%="|"&="|"|="|"^="|"<<="|">>="    { printf("OPERATOR: %s\n", yytext); }

"{"|"}"|"["|"]"|"("|")"|","|";"|"."|":"|"?"   { printf("SPECIAL_SYMBOL: %s\n", yytext); }

.                    { printf("UNRECOGNIZED: %s\n", yytext); }
```

```
%%

int main(int argc, char *argv[]) {
  FILE *file;

  if (argc != 2) {
    printf("Usage: %s filename\n", argv[0]);
    return 1;
  }

  file = fopen(argv[1], "r");
  if (!file) {
    printf("Could not open file %s\n", argv[1]);
    return 1;
  }

  yyin = file;
  yylex();

  fclose(file);
  return 0;
}

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

SAMPLE C PROGRAM:-

```
#include <stdio.h>

/* This is a multi-line comment
   spanning multiple lines */

// This is a single-line comment

#define MAX 100

int main() {
    int num = 42;
    float pi = 3.14159;
    char ch = 'A';
    char* message = "Hello, World!";

    if (num > 0) {
        printf("%s: %d\n", message, num);
    } else {
        return -1;
    }

    for (int i = 0; i < 10; i++) {
        num += i;  // Increment num by i
    }

    int hex_value = 0xABCD;

    return 0;
}
```

6.Program to implement Recursive Descent Parsing in C.

CODE:-

```
#include<stdio.h>
#include<stdlib.h>
/*
E-> iE_
E_-> +iE_ / -iE_ / epsilon
*/
char s[20];
int i=1;
char l;
int match(char t)
{
    if(l==t){
        l=s[i];
        i++; }
    else{
       printf("Sytax error");
       exit(1);}
}

int E_()
{
    if(l=='+'){
        match('+');
        match('i');
        E_(); }
    else  if(l=='-'){
        match('-');
        match('i');
        E_(); }
    else
        return(1);
}

int E()
{
    if(l=='i'){
        match('i');
        E_(); }
}

int main()
{
    printf("\nEnter the set of characters to be checked :");
    scanf("%s",&s);
    l=s[0];
    E();
    if(l=='$')
    {
        printf("Success \n");
    }
    else{
        printf("syntax error");
    }
    return 0;
}
```

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

YACC (Yet Another Compiler Compiler) is a parser generator tool used in compiler design to produce syntax analyzers for context-free grammars. It works closely with Lex, where Lex handles lexical analysis and YACC performs syntax analysis based on grammar rules. YACC generates efficient LALR(1) parsers and allows grammar specification in a structured format with declarations, grammar rules, and associated C actions. It interprets token streams from Lex to understand program structure and syntax. Widely used in building compilers and interpreters, YACC simplifies the implementation of parsing logic for programming languages.

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

Code:

LEX FILE:-

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


%%
[0-9]+              { yylval = atoi(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*  { return IDENTIFIER; }
[+\-*/]             { return yytext[0]; }
\n                { return '\n'; }
[ \t]             { /* skip whitespace */ }
.                 { return 0; }  // Invalid character triggers error in yacc
%%


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

YAAC FILE:-

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

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

%token NUMBER IDENTIFIER

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

%%
stmt: expr '\n' { printf("Valid\n"); }
   ;

expr: expr '+' expr
   | expr '-' expr
   | expr '*' expr
   | expr '/' expr
   | NUMBER
   | IDENTIFIER
   ;
%%

int main() {
   return yyparse();
}

int yyerror(char *s) {
   printf("Invalid\n");
   return 0;
}
```

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

Code:

LEX FILE:-

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

%%
[0-9]+     { yylval = atoi(yytext); return NUMBER; }
[+\-*/\n]   { return yytext[0]; }
[ \t]      { /* ignore whitespace */ }
%%

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

YAAC FILE:-

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

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

%token NUMBER

// Precedence and associativity rules
%left '+' '-'
%left '*' '/'

%%
```

```
stmt: expr '\n' { printf("Result = %d\n", $1); }
  ;

expr: expr '+' expr { $$ = $1 + $3; }
  | expr '-' expr { $$ = $1 - $3; }
  | expr '*' expr { $$ = $1 * $3; }
  | expr '/' expr {
    if ($3 == 0) {
      printf("Error: Divide by zero\n");
      exit(1);
    }
    $$ = $1 / $3;
   }
  | NUMBER
  ;
%%

int main() {
  return yyparse();
}

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

7. d. Create Yacc and Lex specification files are used to convert infix expression to postfix expression.

code:-

<u>LEX FILE:-</u>

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

%%
[0-9]+              { yylval.num = atoi(yytext); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]*  { yylval.id = strdup(yytext); return IDENTIFIER; }
[+\-*/()\n]          { return yytext[0]; }
[ \t]+              { /* ignore whitespace */ }
.                   { printf("Invalid character: %s\n", yytext); return -1; }
%%

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

<u>YAAC FILE:-</u>

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

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

%union {
   int num;
   char* id;
}
```

```
%token <num> NUMBER
%token <id> IDENTIFIER

%%
stmt: expr '\n' { printf("\n"); }
   ;

expr: expr '+' term   { printf("+ "); }
   | expr '-' term   { printf("- "); }
   | term
   ;

term: term '*' factor { printf("* "); }
   | term '/' factor { printf("/ "); }
   | factor
   ;

factor: '(' expr ')'
     | NUMBER       { printf("%d ", $1); }
     | IDENTIFIER    { printf("%s ", $1); free($1); }
     ;
%%

int main() {
   return yyparse();
}

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