

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“Jnana Sangama”, Belgaum -590014, Karnataka.



LAB REPORT on COMPILER DESIGN

Submitted by

SAGAR C MANNANNAVAR(1BM21CS181)

Under the Guidance of

**Dr. Pallavi G B
Assistant Professor,
BMSCE**

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “**Compiler Design**” carried out by **Sagar C Mannannavar(1BM21CS181)** , who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Compiler Design- (22CS5PCCPD)** work prescribed for the said degree.

Dr. Pallavi G B
Assistant professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING



DECLARATION

I, Sagar C Mannannavar(1BM21CS181), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Dr Pallavi G B, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

Lab No	Title	Page No
1		6-8
1.1	Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.	6
1.2	Write a program in LEX to count the number of characters and digits in a string.	7
	Write a program in LEX to count the number of vowels and consonants in a string.	8
2		9-15
2.1	Write a program in lex to count the number of words in a sentence.	9
2.2	Write a program in lex to demonstrate regular definition.	10
2.3	Write a program in lex to identify tokens in a program by taking input from a file and printing the output on the terminal.	11-12
2.4	Write a program in lex to identify tokens in a program by taking input from a file and printing the output in another file.	13-14
2.5	Write a program in lex to find the length of the input string.	15
3		16-23
3.1	Write a program in LEX to recognize Floating Point Numbers.	16
3.2	Read and input sentence, and check if it is compound or simple. If a sentence has the word- and , or ,but ,because ,if ,then ,nevertheless then it is compound else it is simple.	17
3.3	Write a program to check if the input sentence ends with any of the following punctuation marks (? , fullstop , !)	18-19
3.4	Write a program to read an input sentence and to check if the sentence begins with English articles (A, a,AN,An,THE and The).	20-21
3.5	Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt.	22
3.6	Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.	23
4		24-34
4.1	Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.	24-25
4.2	Write a LEX program to recognize the following tokens over the alphabets {0,1,...,9}	26-36

4.2.1	The set of all string ending in 00.	26
4.2.2	The set of all strings with three consecutive 222's.	27
4.2.3	The set of all string such that every block of five consecutive symbols contains at least two 5's.	28
4.2.4	The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.	29
4.2.5	The set of all strings such that the 10th symbol from the right end is 1.	30-31
4.2.6	The set of all four digits numbers whose sum is 9.	31-32
4.2.7	The set of all four digital numbers, whose individual digits are in ascending order from left to right.	33-34
5		35-36
5.1	Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.	35-36
6		37-38
6.1	Write a program to perform recursive descent parsing on the following grammar: S->cAd A->ab a	37-38
7		39-45
7.1	Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.	39-40
7.2	Write a program in YACC to recognize strings of the form $\{(a^n)b, n \geq 5\}$.	41-42
7.3	Write a program in YACC to generate a syntax tree for a given arithmetic expression.	43-45
8		46-47
8.1	Write a program in YACC to convert infix to postfix expression.	46-47
9		48-50
9.1	Write a program in YACC to generate three address code for a given expression.	48-50

Lab 1

1.1 Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.

Code:

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

%}

%%

printf[for|void|main|while|do|switch|case|int|char|float|double|if|else {printf("%s-keyword\n",yytext);
, {printf("%s-separator\n",yytext);}
; {printf("%s-delimiter\n",yytext);}

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

">"|"<"|">="|"<="|"==" {printf("%s- Relational operator\n",yytext);}

"=" {printf("%s-assignment operator\n",yytext);}

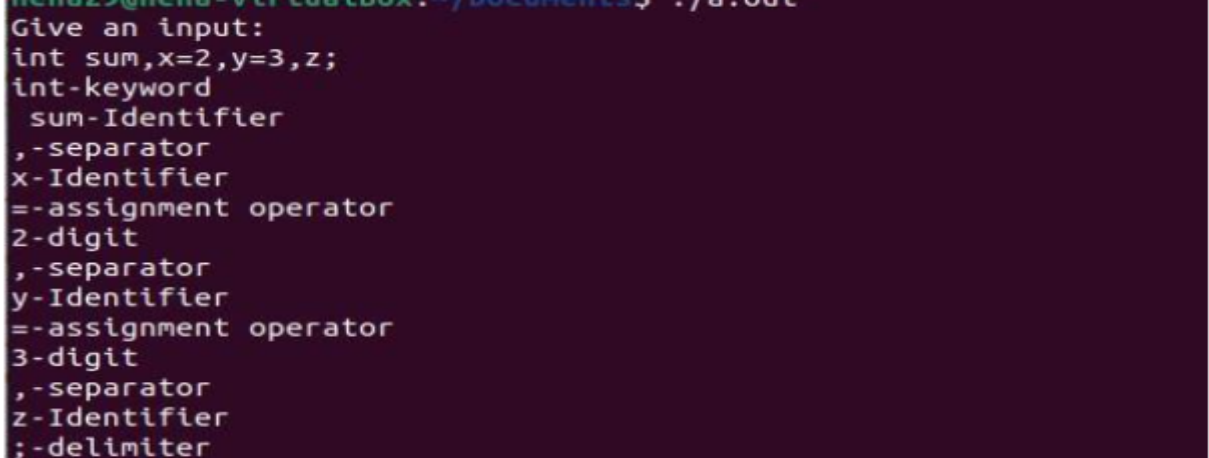
[0-9]+ {printf("%s-digit\n",yytext);}

%%

void main()
{
printf("Give an input:\n");
yylex();
}

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

Output



```
Give an input:
int sum,x=2,y=3,z;
int-keyword
sum-Identifier
,-separator
x-Identifier
=-assignment operator
2-digit
,-separator
y-Identifier
=-assignment operator
3-digit
,-separator
z-Identifier
;-delimiter
```

1.2 Write a program in LEX to count the number of characters and digits in a string. Code

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

int d=0,c=0;

%}

%%

[a-zA-Z] {c++;}
[0-9] {d++;}

. ;

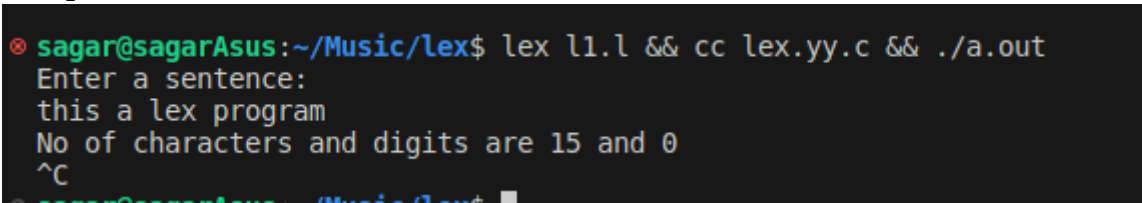
\n {printf("No of characters and digits are %d and %d\n",c,d),c=0,d=0;}

%%

void main()
{
printf("Enter a sentence:\n");
yylex();
}

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

Output



```
sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Enter a sentence:
this a lex program
No of characters and digits are 15 and 0
^C
```

1.3 Write a program in LEX to count the number of vowels and consonants in a string. Code

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

int v=0,c=0;

%}

%%

[AEIOUaeiou]
{v++;} [A-Za-z]
{c++;}

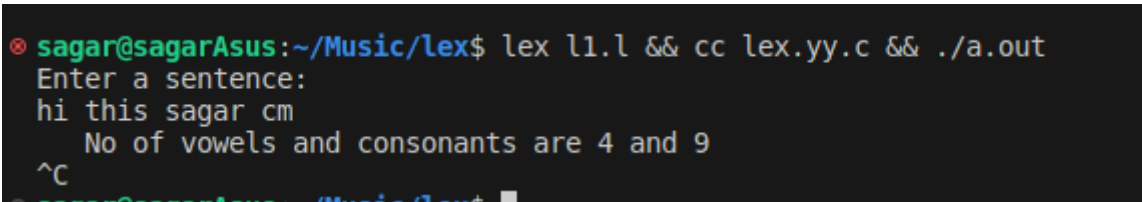
\n {printf("No of vowels and consonants are %d and %d\n",v,c),v=0,c=0;}

%%

void main()
{
printf("Enter a sentence:\n");
yylex();
}

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

Output



```
sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Enter a sentence:
hi this sagar cm
No of vowels and consonants are 4 and 9
^C
```


Lab 2

2.1 Write a program in lex to count the number of words in a sentence. Code

```
%{
#include
<stdio.h
> int
words;
%}
%%

[^\t\n ]+ {words++;}

\n {printf("No of words in the sentence are %d.\n",words),words=0;}

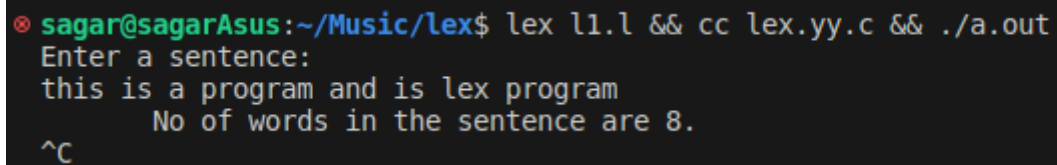
%%

void main()
{
printf("Enter a
sentence:\n");

yylex();
}

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

Output



```
⊗ sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Enter a sentence:
this is a program and is lex program
      No of words in the sentence are 8.
^C
```

2.2 Write a program in lex to demonstrate regular definition.

Code

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

%}

alpha [a-zA-Z0-9]

%%

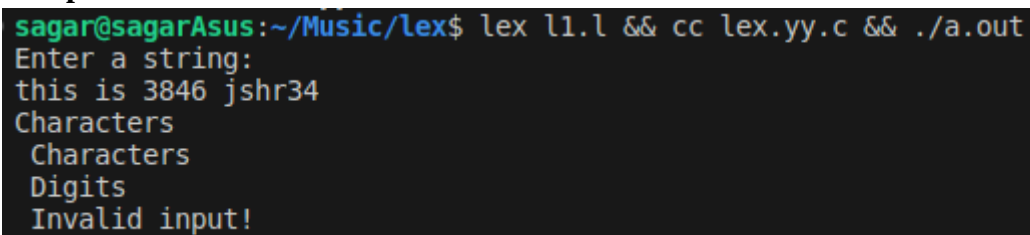
[a-zA-Z]+ {printf("Characters\n");}
[0-9]+ {printf("Digits");}
{alpha}+ {printf("Invalid input!\n");}

%%

void main()
{
printf("Enter a string:\n");
yylex();
}

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

Output



```
sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Enter a string:
this is 3846 jshr34
Characters
Characters
Digits
Invalid input!
```

2.3 Write a program in lex to identify tokens in a program by taking input from a file and printing the output on the terminal.

Code

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

%}

%%

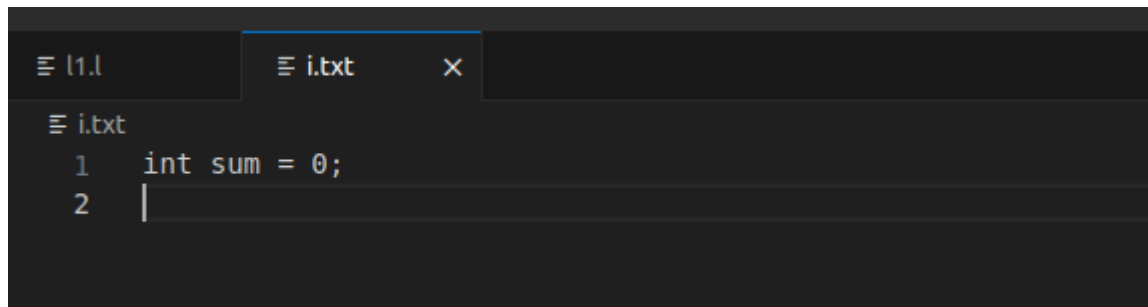
char|int|float {printf("%s is a keyword.\n",yytext);}
[a-zA-Z][a-zA-Z0-9]* {printf("%s is an identifier.\n",yytext);}
, {printf("%s is a separator.\n",yytext);}
; {printf("%s is a delimiter.\n",yytext);}
"=" {printf("%s is an assignment operator.\n",yytext);}
"+"|"-"|"*"|"/" {printf("%s is a binary operator.\n",yytext);}
[0-9]+ {printf("%s is/are digit(s).\n",yytext);}
\n ;

%%

void main()
{
yyin=fopen("i.txt","r");
yylex();
fclose(yyin);
}

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

Output



The screenshot shows a text editor window with two tabs: 'l1.l' and 'i.txt'. The 'i.txt' tab is active. The content of 'i.txt' is as follows:

```
≡ i.txt
1  int sum = 0;
2  |
```

```
● sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
int is a keyword.
sum is an identifier.
= is an assignment operator.
0 is/are digit(s).
; is a delimiter.
○ sagar@sagarAsus:~/Music/lex$
```

2.4 Write a program in lex to identify tokens in a program by taking input from a file and printing the output in another file.

Code

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

%}

%%

char|int|float {fprintf(yyout,"%s is a keyword.\n",yytext);}
[a-zA-Z][a-zA-Z0-9]* {fprintf(yyout,"%s is an identifier.\n",yytext);}
, {fprintf(yyout,"%s is a separator.\n",yytext);}
; {fprintf(yyout,"%s is a delimiter.\n",yytext);}
"=" {fprintf(yyout,"%s is an assignment operator.\n",yytext);}
"+"|"-"|"*"|"|" "/" {fprintf(yyout,"%s is a binary operator.\n",yytext);}
[0-9]+ {fprintf(yyout,"%s is/are digit(s).\n",yytext);}

\n ;

%%

void main()
{
yyin=fopen("i.txt","r");
yyout=fopen("output.txt","w");
yylex();
printf("Printed in output.txt\n");
fclose(yyin);
fclose(yyout);
}

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

Output

```
≡ l1.l  ≡ i.txt  ×
≡ i.txt
1  int sum = 0;
2  |
```

```
● sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Printed in output.txt
○ sagar@sagarAsus:~/Music/lex$
```

```
≡ l1.l  ≡ output.txt  ×  ≡ i.txt
≡ output.txt
1  int is a keyword.
2  sum is an identifier.
3  = is an assignment operator.
4  0 is/are digit(s).
5  ; is a delimiter.
6  |
```

2.5 Write a program in lex to find the length of the input string. Code

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

%}

%%

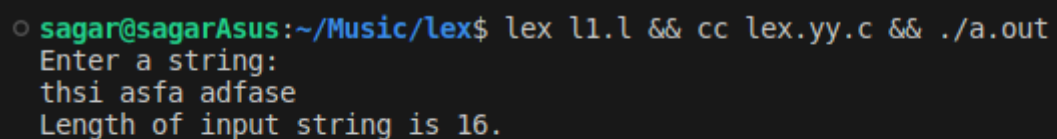
[a-zA-Z0-9.,!? \t]+ {printf("Length of input string is %d.\n",yyleng);}

%%

void main()
{
printf("Enter a string:\n");
yylex();
}

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

Output



```
○ sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Enter a string:
thsi asfa adfase
Length of input string is 16.
█
```

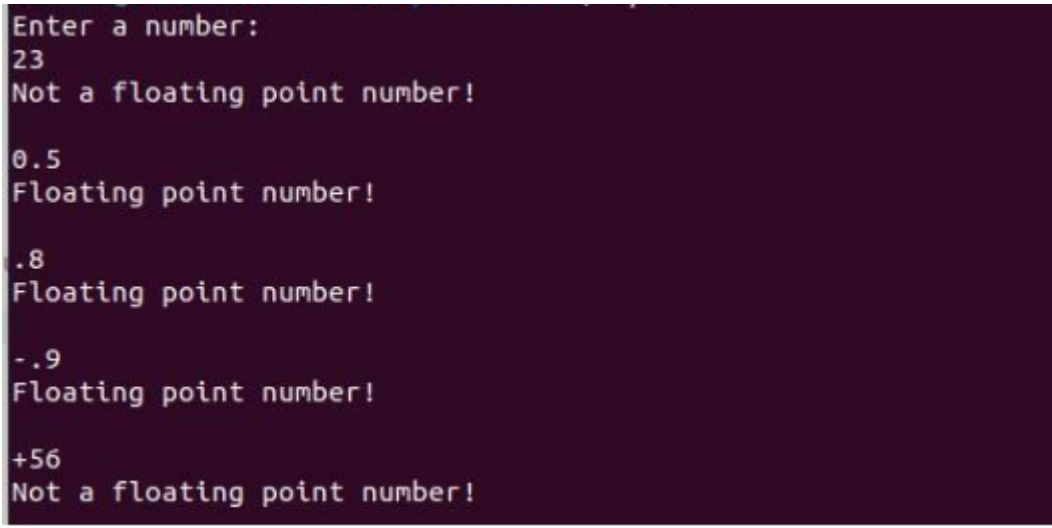
Lab 3

3.1 Write a program in LEX to recognize Floating Point

Numbers. Code

```
%{  
#include<stdio.h>  
%}  
%%  
[+-]?[0-9]*[.][0-9][0-9]* {printf("Floating point number!\n");};  
[+-]?[0-9][0-9]* {printf("Not a floating point number!\n");};  
%%  
int yywrap()  
{  
return 1;  
}  
void main()  
{  
printf("Enter a number:\n");  
yylex();  
}
```

Output



```
Enter a number:  
23  
Not a floating point number!  
  
0.5  
Floating point number!  
  
.8  
Floating point number!  
  
-.9  
Floating point number!  
  
+56  
Not a floating point number!
```


3.2 Read and input sentence, and check if it is compound or simple. If a sentence has the word- and , or ,but ,because ,if ,then ,nevertheless then it is compound else it is simple.

Code

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

int flag=0;

%}

%%

if|then|but|because|nevertheless|and|or {flag=1;}

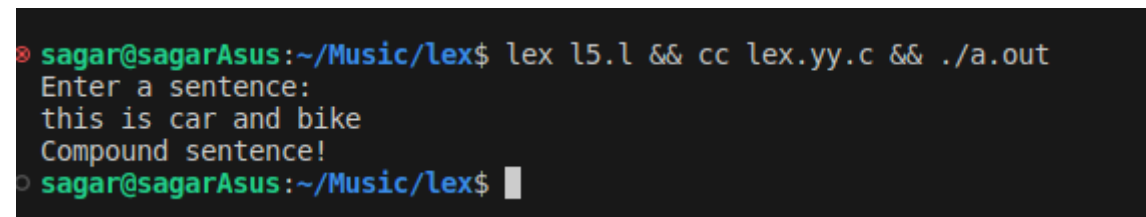
.;
\n {return 0;}

%%

int yywrap()
{
return 1;
}

void main()
{
printf("Enter a sentence:\n");
yylex();
if(flag==1)
printf("Compound sentence!\n");
else
printf("Simple sentence!\n");
}
```

Output



```
sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out
Enter a sentence:
this is car and bike
Compound sentence!
sagar@sagarAsus:~/Music/lex$
```

3.3 Write a program to check if the input sentence ends with any of the following punctuation marks (? , fullstop , !)

Code

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

int flag=0;

%}

%%

.*[?!|.]$ {flag=1;}

.* {flag=0;}

\n {return 0;}

%%

int yywrap()

{
return 1;
}

void main()

{
printf("Enter a sentence:\n");
yylex();
if(flag==1)
printf("Ends with a punctuation!\n");
else
printf("Does not end with punctuation!\n");
}
```

Output

```
sagar@sagarAsus:~/Music/lex$ lex l1.l && cc lex.yy.c && ./a.out
Enter a sentence:
this is. ,sdfa ?
Ends with a punctuation!
sagar@sagarAsus:~/Music/lex$
```

3.4 Write a program to read an input sentence and to check if the sentence begins with English articles (A, a, AN, An, THE and The).

Code

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

int flag=0;

%}

%%

^(an|An|The|the|A|a)[ " ].* {flag=1;}

.* {flag=0;}

\n {return 0;}

%%

int yywrap()
{
return 1;
}

void main()
{
printf("Enter a sentence:\n");
yylex();
if(flag==1)
printf("Starts with an article!\n");
else
printf("Does not start with an article!\n");
}
```

Output

```
sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out
Enter a sentence:
the bike is a bike
Starts with an article!
sagar@sagarAsus:~/Music/lex$
```

3.5 Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt.

Code

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

int c=0;

%}

%%

"\\"[^\"]*\\*+([\/*][^\"]*\\*+)*\ {c++;}

"/".* {c++;}

. ECHO;

%%

int yywrap()
{
return 1;
}

void main()
{
yyin=fopen("input.txt","r");
yyout=fopen("output.txt","w");
yylex();

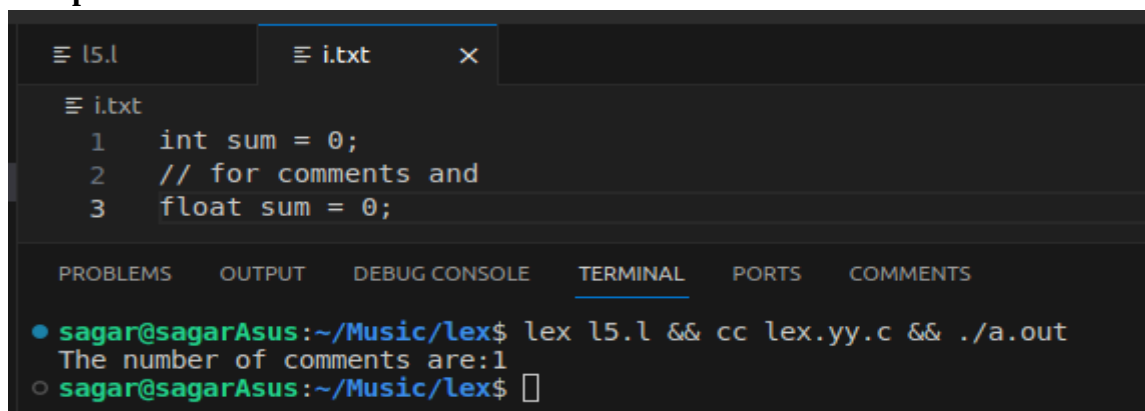
printf("The number of comments are:%d\n",c);

fclose(yyin);

fclose(yyout);

}
```

Output



The screenshot shows a code editor with two tabs: 'l5.1' and 'i.txt'. The 'i.txt' tab is active, displaying the following code:

```
1 int sum = 0;
2 // for comments and
3 float sum = 0;
```

Below the code editor is a terminal window with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'COMMENTS'. The 'TERMINAL' tab is active, showing the following commands and output:

```
sagar@sagarAsus:~/Music/lex$ lex l5.1 && cc lex.yy.c && ./a.out
The number of comments are:1
sagar@sagarAsus:~/Music/lex$
```

3.6 Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.

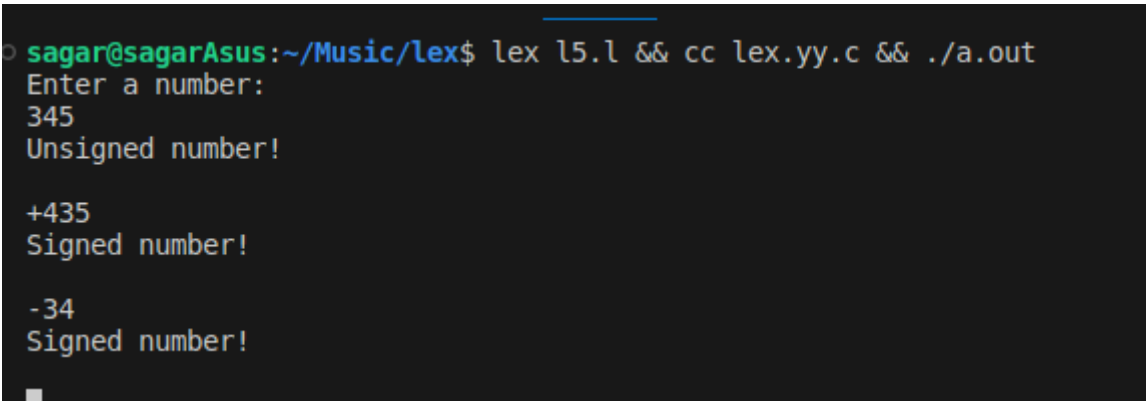
Code

```
%{
#include<stdio.h>
%}
%%
[+|-][0-9]+ {printf("Signed number!\n");}
[0-9]+ {printf("Unsigned number!\n");}
%%

int yywrap()
{
return 1;
}

void main()
{
printf("Enter a number:\n");
yylex();
}
```

Output



```
sagar@sagarAsus:~/Music/lex$ lex l5.1 && cc lex.yy.c && ./a.out
Enter a number:
345
Unsigned number!

+435
Signed number!

-34
Signed number!
```

Lab 4

4.1 Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.

Code

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

%}

%%

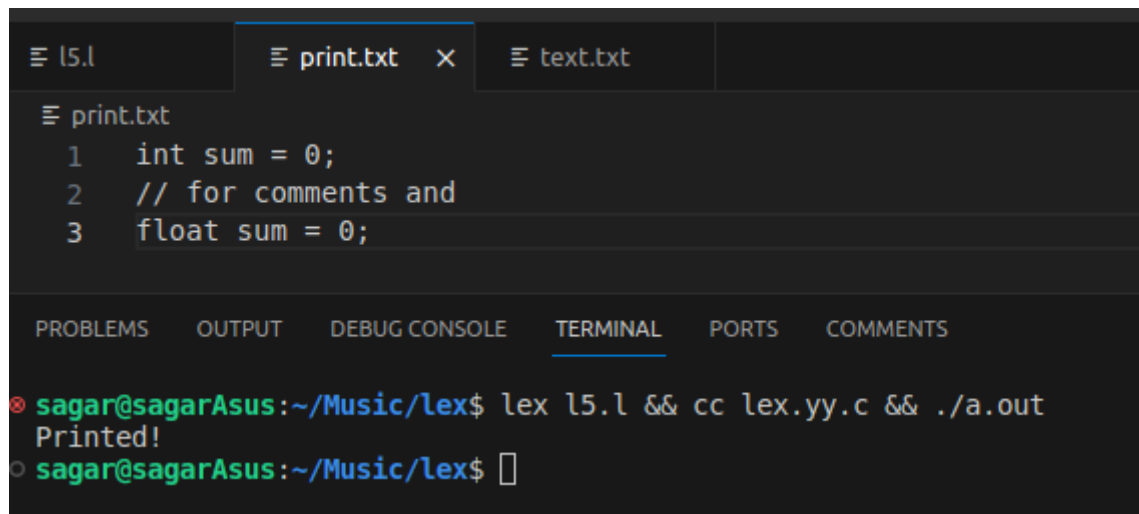
[ \t]+ {fprintf(yyout," ");}
.\n {fprintf(yyout,"%s",yytext);}

%%

void main()
{
yyin=fopen("text.txt","r");
yyout=fopen("print.txt","w");
yylex();
fclose(yyin);
fclose(yyout);
printf("Printed!\n");
}

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


Output



The image shows a code editor with three tabs: `l5.l`, `print.txt`, and `text.txt`. The `print.txt` tab is active, displaying the following code:

```
1 int sum = 0;  
2 // for comments and  
3 float sum = 0;
```

Below the code editor is a terminal window with tabs for `PROBLEMS`, `OUTPUT`, `DEBUG CONSOLE`, `TERMINAL`, `PORTS`, and `COMMENTS`. The `TERMINAL` tab is active, showing the following commands and output:

```
sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out  
Printed!  
sagar@sagarAsus:~/Music/lex$
```

4.2 Write a LEX program to recognize the following tokens over the alphabets {0,1,...,9}

4.2.1 The set of all string ending in

00. Code

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

int flag=0;

%}

%%

[0-9]+[00] {flag=1;}

. ;

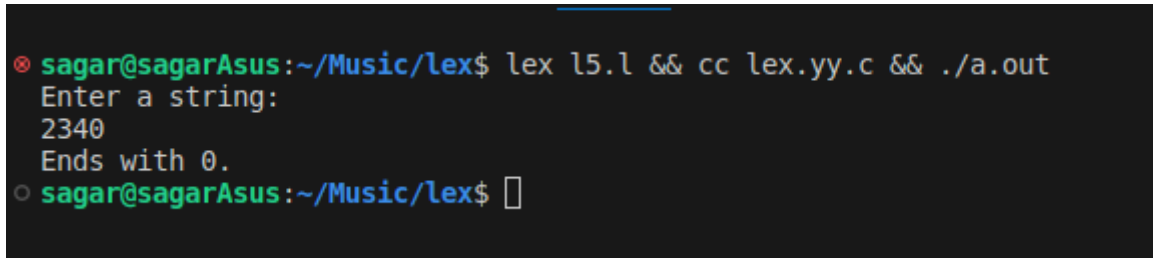
\n {return 0;}

%%

void main()
{
printf("Enter a string:\n");
yylex();
if(flag==1)
printf("Ends with 0.\n");
else
printf("Does not end with 0.\n");
}

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

Output



```
✖ sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out
Enter a string:
2340
Ends with 0.
○ sagar@sagarAsus:~/Music/lex$
```

4.2.2 The set of all strings with three consecutive

222's. Code

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

int flag=0;

%}

%%

[0-9]*[2][2][2][0-9]* {flag=1;}

. ;

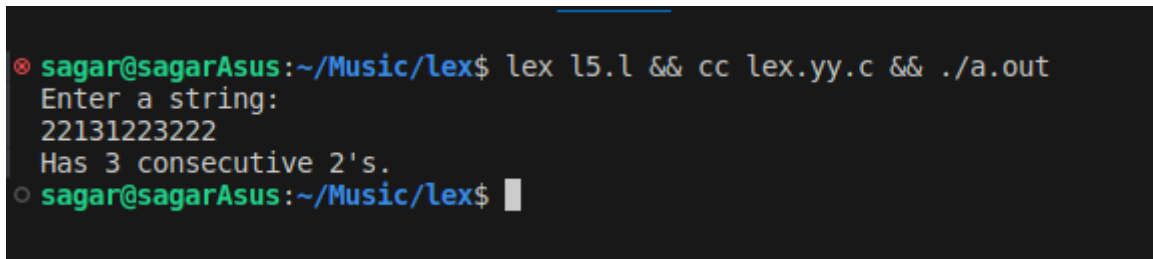
\n {return 0;}

%%

void main()
{
printf("Enter a string:\n");
yylex();
if(flag==1)
printf("Has 3 consecutive 2's.\n");
else
printf("Does not have 3 consecutive 2's.\n");
}

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

Output



```
⊗ sagar@sagarAsus:~/Music/lex$ lex l5.1 && cc lex.yy.c && ./a.out
Enter a string:
22131223222
Has 3 consecutive 2's.
○ sagar@sagarAsus:~/Music/lex$
```

4.2.3 The set of all string such that every block of five consecutive symbols contains at least two 5's.

Code

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

int i,count=0,flag;

%}

%%

.{1,5} {flag=0;
for(i=0;i<5;i++)
{
    int c=yytext[i]-'0';
    if(c==5)
    {
        count++;
        if(count==2)
        {
            flag=1;
            break;
        }
    }
}
count=0;

printf("yytext:%s,flag(1 if no of 5 is atleast 2):%d\n",yytext,flag);
if(flag!=1)
{
    printf("Not a valid string!\n");
    return 0;
}
}

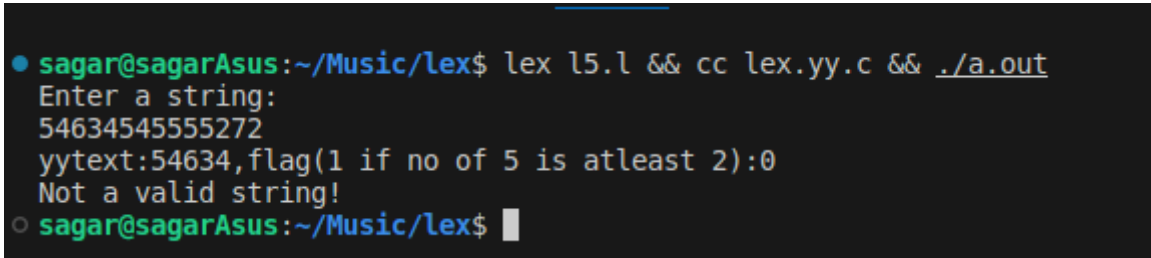
\n {return 0;}

%%

void main()
```

```
{  
printf("Enter a string:\n");  
yylex();  
if(flag==1)  
printf("Valid string.\n");  
}  
int yywrap()  
{  
return 1;  
}
```

Output



```
● sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out  
Enter a string:  
54634545555272  
yytext:54634,flag(1 if no of 5 is atleast 2):0  
Not a valid string!  
○ sagar@sagarAsus:~/Music/lex$
```

4.2.4 The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.

Code

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

int c,i,flag=1,sum=0,power=1;
%}

%%

^1[01]* {for(i=yy leng-1;i>=0;i--)
    {
        c=yytext[i]-'0';
        sum+=c*power;
        power*=2;
    }
    printf("Decimal representation:%d\n",sum);
    if(sum%5!=0)
    {
        printf("Not congruent to modulo 5.\n");
        sum=0;
        power=1;
    }
    else
    {
        printf("Congruent to modulo 5.\n");
        sum=0;
        power=1;
    }
}

.* {printf("Not a binary number.\n");}

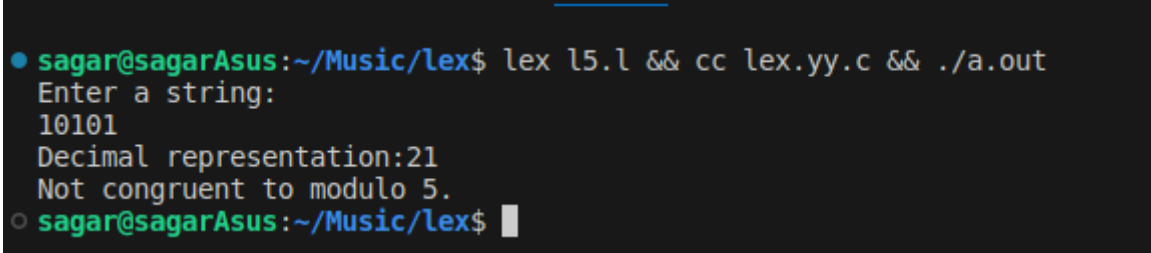
\n {return 0;}

%%

void main()
{
    printf("Enter a string:\n");
```

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

Output



```
● sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out  
Enter a string:  
10101  
Decimal representation:21  
Not congruent to modulo 5.  
○ sagar@sagarAsus:~/Music/lex$
```

4.2.5 The set of all strings such that the 10th symbol from the right end is

1. Code

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

int flag=0;

%}

%%

[0-9]*1[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9] {flag=1;}

.;

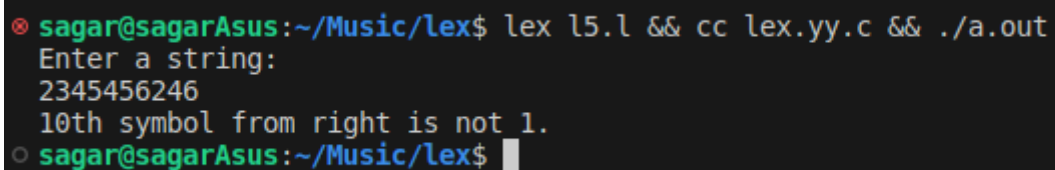
\n {return 0;}

%%

void main()
{
printf("Enter a string:\n");
yylex();
if(flag==1)
printf("10th symbol from right is 1.\n");
else
printf("10th symbol from right is not 1.\n");
}

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

Output



```
✖ sagar@sagarAsus:~/Music/lex$ lex l5.l && cc lex.yy.c && ./a.out
Enter a string:
2345456246
10th symbol from right is not 1.
○ sagar@sagarAsus:~/Music/lex$
```


4.2.6 The set of all four digits numbers whose sum is

9. Code

```
%{  
#include<stdio.h>  
int sum=0,i,flag=0;  
%}  
%%  
[0-9][0-9][0-9][0-9] {for(i=0;i<yyleng;i++)  
    {  
        sum+=yytext[i]-'0';  
    }  
    if(sum==9)  
    {  
        flag=1;  
        sum=0;  
    }  
    else  
    {  
        flag=0;  
        sum=0;  
    }  
}  
\n {return 0;}  
%%  
void main()  
{  
    printf("Enter a string:\n");  
    yylex();  
    if(flag==1)  
        printf("The sum of digits is 9.\n");  
    else  
        printf("The sum of digits is not 9.\n");  
}  
int yywrap()
```

```
{  
return 1;  
}
```

Lab 5

Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.

Code

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

void lexicalAnalyzer(char input_code[]) {
    char *keywords[] = {"if", "else", "while", "for", "return"};
    char *operators[] = {"+", "-", "*", "/", "=", "==", "<", ">", "<=", ">="};
    char *punctuations[] = {"", ";", "(", ")", "{", "}"};

    char *token = strtok(input_code, " \t\n");

    while (token != NULL) {
        if (isdigit(token[0])) {
            printf("Number: %s\n", token);
        } else if (isalpha(token[0]) || token[0] == '_' ) {
            int isKeyword = 0;
            for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {
                if (strcmp(token, keywords[i]) == 0) {
                    printf("Keyword: %s\n", token);
                    isKeyword = 1;
                    break;
                }
            }
            if (!isKeyword) {
                printf("Identifier: %s\n", token);
            }
        } else if (strchr("+-*/=<>(){}[]", token[0]) != NULL) {
            printf("Operator: %s\n", token);
        }
        else if (strchr(";", token[0]) != NULL)
```

```

    {
        printf("Punctuation:%s\n",token);
    }

    token = strtok(NULL, " \t\n");
}
}

int main() {
    char input_code[] = "if ( x > 0 ) { return x ; } else { return -x ; }";
    lexicalAnalyzer(input_code);
    return 0;
}

```

Output

```

PS C:\Users\neha2\OneDrive\Documents\CD_lab_Practice> cd
lysis } ; if ($?) { .\Week5_lexicalAnalysis }
Keyword: if
Operator: (
Identifier: x
Operator: >
Number: 0
Operator: )
Operator: {
Keyword: return
Identifier: x
Punctuation;;
Operator: }
Keyword: else
Operator: {
Keyword: return
Operator: -x
Punctuation;;
Operator: }

```

Lab 6

Write a program to perform recursive descent parsing on the following grammar:

S->cAd

A->ab | a

Code

```
#include <stdio.h>
#include<stdlib.h>
> char input[100];
int ind = 0;
void match(char expected)
{
    if (input[ind] == expected)
    {
        ind++;
    }
}
void A();
void S()
{
    match('c');
    A();
    match('d');
}
void A()
{
    if (input[ind] == 'a')
    {
        printf("Hello\n");
        match('a');
        match('b');
    } /*else if (input[ind] == 'a')
    {
        printf("Hi!\n");
```

```

        match('a');
    }*/
else
{
    printf("Parsing failed.\n", ind);
    exit(1);
}
}

int main() {
    printf("Enter the input string:\n");
    scanf("%s", input);

    S();

    if (input[ind] == '$') {
        printf("Parsing successful.\n");
    } else {
        printf("Parsing failed. Extra characters found.\n");
    }

    return 0;
}

```

Output

```

PS C:\Users\neha2\OneDrive\Documents\CD_lab_Practice> cd
Descent } ; if ($?) { .\Week6_RecursiveDescent }
Enter a string:
cad$
Valid string!
PS C:\Users\neha2\OneDrive\Documents\CD_lab_Practice> cd
Descent } ; if ($?) { .\Week6_RecursiveDescent }
Enter a string:
caad$
Invalid String!
PS C:\Users\neha2\OneDrive\Documents\CD_lab_Practice> cd
Descent } ; if ($?) { .\Week6_RecursiveDescent }
Enter a string:
cabd$
Valid string!

```

Lab 7

7.1 Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.

Code

LEX

```
%{  
#include<stdio.h>  
#include<stdlib.h>  
#include "y.tab.h"  
extern int yylval;  
%}  
%%  
[0-9]+ {yylval=atoi(yytext);return num;}  
[t ] ;  
\n {return 0;}  
. {return yytext[0];}  
%%  
int yywrap()  
{  
}
```

YACC

```
%{  
#include<stdio.h>  
#include<stdlib.h>  
int yyerror(const char *s);  
int yylex(void);  
%}  
%token num;  
%left '+' '-'  
%left '*' '/'  
%left ')'  
%left '('
```

```

%%

s:e {printf("Valid expression!\n");

    printf("Result:%d\n",$$);

    exit(0);

}

;

e:e'+e {$$=$1+$3;}

|e'-e {$$=$1-$3;}

|e'*e {$$=$1*$3;}

|e'/e {$$=$1/$3;}

|('(e)' {$$=$2;}

|num {$$=$1;}

;

%%

void main()

{

printf("Enter an arithmetic expression:\n");

yyparse();

}

int yyerror(const char *s)

{

printf("Invalid expression!\n");

return 0;

}

```

Output

```

● sagar@sagarAsus:~/Music/lex$ lex l5.l
● sagar@sagarAsus:~/Music/lex$ yacc -d l.y
● sagar@sagarAsus:~/Music/lex$ gcc lex.yy.c y.tab.c
● sagar@sagarAsus:~/Music/lex$ ./a.out
Enter an arithmetic expression:
2-3/5
Valid expression!
Result:2
○ sagar@sagarAsus:~/Music/lex$ █

```


7.2 Write a program in YACC to recognize strings of the form $\{(a^n)b, n \geq 5\}$. Code

LEX

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
}%
%%
[aA] {yylval=yytext[0];return
A;} [bB]
{yylval=yytext[0];return B;}
\n {return NL;}
. {return yytext[0];}
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
#include<stdio.h>
#include<stdlib.h>
int yyerror(char *s);
int yylex(void);
}%
%token A
%token B
%token NL
%%
smtr:A A A A A S B NL {printf("Parsed using the rule  $(a^n)b, n \geq 5$ . Valid String!\n");}
;
S:S A
```

```

};

%%

void main()
{
printf("Enter a string!\n");
yyparse();
}

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

```

Output

```

sagar@sagarAsus:~/Music/lex$ lex l5.l
sagar@sagarAsus:~/Music/lex$ yacc -d l.y
sagar@sagarAsus:~/Music/lex$ gcc lex.yy.c y.tab.c
l.y:2:19: warning: extra tokens at end of #include directive
  2 | #include<stdio.h> #include<stdlib.h> int yyerror(char *s); int yylex(void);
    |                   ^
y.tab.c: In function 'yyparse':
y.tab.c:1021:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1021 |     yychar = yylex ();
    |              ^~~~~
y.tab.c:1162:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'?
1162 |     yyerror (YY_("syntax error"));
    |     ^~~~~~
    |     yyerrok
sagar@sagarAsus:~/Music/lex$ ./a.out
Enter a string!
sreg
Invalid String!

```

7.3 Write a program in YACC to generate syntax tree for a given arithmetic expression. Code

LEX

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern int yyval;
}%
%%
[0-9]+ {yyval=atoi(yytext);return digit;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int yyerror(char *s);
int yylex(void);
struct tree_node
{
char val[10];
int lc;
int rc;
};
```

```

int ind;

struct tree_node syn_tree[100];

void my_print_tree(int cur_ind);

int mknode(int lc,int rc,char
*val);

%}

%token digit

%%

S:E {my_print_tree($1);}

;

E:E+'T' {$$=mknode($1,$3,"+");}

|T {$$=$1;}

;

T:T'*'F {$$= mknode($1,$3,"*");}

|F {$$=$1;}

;

F:'(E)' {$$=$2;}

|digit {char buf[10];sprintf(buf,"%d", yylval);$$ = mknode(-1,-1,buf);}

;

%%

int main()

{

ind=0;

printf("Enter an expression:\n");

yyparse();

return 0;

}

int yyerror(char *s)

{

printf("NITW Error\n");

return 0;

}

int mknode(int lc,int rc,char val[10])

{

strcpy(syn_tree[ind].val,val);

```

```

syn_tree[ind].lc = lc;
syn_tree[ind].rc  =
rc; ind++;
return ind-1;
}

/*my_print_tree function to print the syntax tree in DLR fashion*/
void my_print_tree(int cur_ind)
{
if(cur_ind== -1) return;
if(syn_tree[cur_ind].lc== -1 && syn_tree[cur_ind].rc== -1)
printf("Digit Node -> Index : %d, Value : %s\n", cur_ind, syn_tree[cur_ind].val);
else
printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d, Right Child Index : %d\n", cur_ind, syn_tree[cur_ind].val, syn_tree[cur_ind].lc, syn_tree[cur_ind].rc);
my_print_tree(syn_tree[cur_ind].lc);
my_print_tree(syn_tree[cur_ind].rc);
}

```

Output

```

sagar@sagarAsus:~/Music/lex$ lex l5.l
sagar@sagarAsus:~/Music/lex$ yacc -d l.y
sagar@sagarAsus:~/Music/lex$ gcc lex.yy.c y.tab.c
sagar@sagarAsus:~/Music/lex$ ./a.out
Enter an expression:
4/2+2*2
Digit Node -> Index : 0, Value : 4
NITW Error
sagar@sagarAsus:~/Music/lex$ 

```

Lab 8

8.1 Write a program in YACC to convert infix to postfix expression.

Code

LEX

```
%{  
  
#include<stdio.h>  
  
#include<stdlib.h>  
  
#include "y.tab.h"  
  
extern int yylval;  
  
%}  
  
%%  
  
[0-9]+ {yylval=atoi(yytext);return num;}  
  
[t ] ;  
  
\n {return 0;}  
  
. {return yytext[0];}  
  
%%  
  
int yywrap()  
  
{  
  
}
```

YACC

```
%{  
  
#include<stdio.h>  
  
#include<stdlib.h>  
  
int yyerror(const char *s);  
  
int yylex(void);  
  
%}  
  
%token num  
  
%left '+' '-'  
  
%left '*' '/'  
  
%left ')'   
  
%left '('  
  
%right '^'  
  
%%
```

```

s:e {printf("\n");}

;
e:e'+t {printf("+");}
|e-'t {printf("-");}
|t
;
t:t'*h {printf("*");}
|t/'h {printf("/");}
|h
;
h:f^'h {printf("^");}
|f
;
f:'(e)'
|num {printf("%d", $1);}
;
%%

void main()
{
printf("Enter an infix expression:\n");
yyparse();
}

int yyerror(const char *s)
{
printf("Invalid infix expression!\n");
return 0;
}

```

Output

```

● sagar@sagarAsus:~/Music/lex$ lex l5.l
● sagar@sagarAsus:~/Music/lex$ yacc -d l.y
● sagar@sagarAsus:~/Music/lex$ gcc lex.yy.c y.tab.c
l.y:2:19: warning: extra tokens at end of #include directive
    2 | #include<stdio.h> #include<stdlib.h>
      |                   ^
● sagar@sagarAsus:~/Music/lex$ ./a.out
Enter an infix expression:
2+3*4
234*+
○ sagar@sagarAsus:~/Music/lex$ █

```

Lab 9

9.1 Write a program in YACC to generate three address code for a given expression.

Code

LEX

```
%{  
#include<stdio.h>  
#include<stdlib.h>  
#include"y.tab.h"  
extern int yylval;  
extern char iden[20];  
%}  
d [0-9]+  
a [a-zA-Z]+  
%%  
{d} { yylval=atoi(yytext); return digit; }  
{a} { strcpy(iden,yytext); yylval=1; return id;}  
[ \t] {;}  
\n return 0;  
. return yytext[0];  
%%  
int yywrap()  
{  
return 1;  
}
```

YACC

```
%{  
#include <math.h>  
#include<ctype.h>  
#include<stdio.h>  
int yyerror(char *s);  
int yylex(void);  
int var_cnt=0;  
char iden[20];
```



```

%}

%token id

%token digit

%%

S:id '=' E {printf("%s=t%d\n",iden,var_cnt-1);}

E:E '+' T {$$=var_cnt; var_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 );}

|E '-' T { $$=var_cnt; var_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 );}

|T {$$=$1;}

;

T:T '*' F {$$=var_cnt; var_cnt++; printf("t%d = t%d * t%d;\n", $$, $1, $3 );}

|T '/' F {$$=var_cnt; var_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 );}

|F {$$=$1;}

;

F:P '^' F {$$=var_cnt; var_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );}

|P {$$ = $1;}

;

P: '(' E ')' {$$=$2;}

|digit {$$=var_cnt; var_cnt++; printf("t%d = %d;\n",$$,$1);}

;

%%

int main()

{

var_cnt=0;

printf("Enter an expression:\n");

yyparse();

return 0;

}

int yyerror(char *s)

{

printf("Invalid expression!");

return 0;

}

```

Output

```
● sagar@sagarAsus:~/Music/lex$ lex l5.l
● sagar@sagarAsus:~/Music/lex$ yacc -d l.y
● sagar@sagarAsus:~/Music/lex$ gcc lex.yy.c y.tab.c
● sagar@sagarAsus:~/Music/lex$ ./a.out
Enter an expression:
a=2
t0 = 2;
a=t0
● sagar@sagarAsus:~/Music/lex$ ./a.out
Enter an expression:
a=3+4
t0 = 3;
t1 = 4;
t2 = t0 + t1;
a=t2
```