

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Compiler Design Lab

Submitted by:

Sanchay Agrawal (1BM21CS186)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



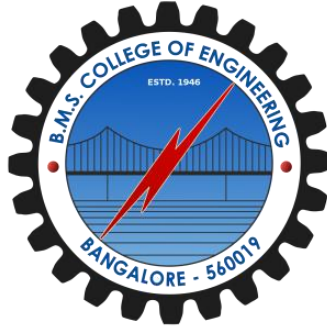
B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Nov 2023 - March 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 **Sanchay Agrawal (1BM21CS186)**, who is 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 GB
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Table Of Contents

S.No.	Experiment Title		Page No.
1	Course Outcomes		1
2	Experiments		1 - 59
	2.1	Experiment - 1	1
	2.1.1	Question: LEX program to check whether the entered user input is a number, operator, or invalid character.	1
	2.1.2	Code	1
	2.1.3	Output	1
	2.2	Experiment - 2	2
	2.2.1	Question: LEX program to perform word count on the entered user input.	2
	2.2.2	Code	2
	2.2.3	Output	2
	2.3	Experiment - 3	3
	2.3.1	Question: LEX program to perform vowel count and consonant count.	3
	2.3.2	Code	3
	2.3.3	Output	3
	2.4	Experiment - 4	4
	2.4.1	Question: LEX program to identify whether the entered user input is a keyword, separator or identifier.	4
	2.4.2	Code	4
	2.4.3	Output	4
	2.5	Experiment - 5	5
	2.5.1	Question: LEX program to print the same input that the user	5

			entered again using ECHO.	
		2.5.2	Code	5
		2.5.3	Output	5
	2.6	Experiment - 6		6
		2.6.1	Question: LEX program to check whether input is digit or not.	6
		2.6.2	Code	6
		2.6.3	Output	6
	2.7	Experiment - 7		7
		2.7.1	Question: LEX program to check whether the given number is even or odd.	7
		2.7.2	Code	7
		2.7.3	Output	7
	2.8	Experiment - 8		8 - 9
		2.8.1	Question: LEX program to check whether a number is prime or not.	8
		2.8.2	Code	8
		2.8.3	Output	9
	2.9	Experiment - 9		10
		2.9.1	Question: LEX program to print invalid string if a alpha numeric string is entered as an input.	10
		2.9.2	Code	10
		2.9.3	Output	10
	2.10	Experiment - 10		11
		2.10.1	Question: LEX program to recognize identifiers, keyword (int and float), anything else as invalid tokens.	11

		2.10.2	Code	11
		2.10.3	Output	11
	2.11	Experiment - 11		12
		2.11.1	Question: LEX program to recognize identifiers, keyword (int and float), anything else as invalid tokens. Read these from a text file.	12
		2.11.2	Code	12
		2.11.3	Output	12
	2.12	Experiment - 12		13 - 14
		2.12.1	Question: (a) LEX program to identify alphabets as characters and numbers as digits. Expected Output: Enter input: sum33 Sum is a stream of characters 33 is a digit (b) Modify the above program to get the following output: Enter input: sum33 s is a character u is a character m is a character 3 is a digit 3 is a digit	13
		2.12.2	Code	13
		2.12.3	Output	14
	2.13	Experiment - 13		15
		2.13.1	Question: 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 sentence else it is simple sentence.	15
		2.13.2	Code	15
		2.13.3	Output	15

	2.14	Experiment - 14		16
	2.14.1	Question: 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). If sentence starts with the article, appropriate message should be printed. If sentence does not start with the article, appropriate message should be printed.		16
	2.14.2	Code		16
	2.14.3	Output		16
	2.15	Experiment - 15		17
	2.15.1	Question: Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.		17
	2.15.2	Code		17
	2.15.3	Output		17
	2.16	Experiment - 16		18
	2.16.1	Question: Write a program to check if the input sentence ends with any of the following punctuation marks (? , fullstop , !)		18
	2.16.2	Code		18
	2.16.3	Output		18
	2.17	Experiment - 17		19
	2.17.1	Question: 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.		19
	2.17.2	Code		19
	2.17.3	Output		19

	2.18	Experiment - 18	20
	2.18.1	Question: Write a program in LEX to recognize Floating Point Numbers.	20
	2.18.2	Code	20
	2.18.3	Output	20
	2.19	Experiment - 19	21
	2.19.1	Question: Write a program in LEX to recognize different tokens: Keywords, identifiers, Constants, Operators and Punctuation symbols.	21
	2.19.2	Code	21
	2.19.3	Output	21
	2.20	Experiment - 20	22
	2.20.1	Question: Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.	22
	2.20.2	Code	22
	2.20.3	Output	22
	2.21	Experiment - 21	23 - 29
	2.21.1	Question: Write a LEX program to recognize the following tokens over the digits [0-9]: a) The set of all string ending in 00. b) The set of all strings with three consecutive 222's. c) The set of all string such that every block of five consecutive symbols contains at least two 5's. d) The set of all strings beginning with a 1 which, interpreted as the binary representation of an	23

		<p>integer, is congruent to zero modulo 5.</p> <p>e) The set of all strings such that the 10th symbol from the right end is 1.</p> <p>f) The set of all four digits numbers whose sum is 9.</p> <p>g) The set of all four digital numbers, whose individual digits are in ascending order from left to right.</p>	
	2.21.2	Code	29
	2.21.3	Output	28
2.22	Experiment - 22		30 - 31
	2.22.1	<p>Question:</p> <p>Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuation.</p>	30
	2.22.2	Code	30
	2.22.3	Output	31
2.23	Experiment - 23		32 - 34
	2.23.1	<p>Question:</p> <p>a) Design a suitable grammar for evaluation of arithmetic expression having + and – operators. + has least priority and it is left associative - has higher priority and is right associative</p> <p>b) Design a suitable grammar for evaluation of arithmetic expression having + , – , * , / , %, ^ operators. ^ having highest priority and right associative % having second highest priority and left associative * , / have third highest priority and left associative + , - having least priority and left associative</p>	32
	2.23.2	Code	32
	2.23.3	Output	34

	2.24	Experiment - 24	35 - 40
	2.24.1	Question: a) Use YACC to generate Syntax tree for a given expression. b) Update the above program based on the following priority and associativity: + , - having least priority and left associative * , / have second highest priority and left associative ^ having next highest priority and right associative () having highest priority	35
	2.24.2	Code	35
	2.24.3	Output	40
	2.25	Experiment - 25	41 - 44
	2.25.1	Question: a) Use YACC to convert: Infix expression to Postfix expression. b) Modify the above program so as to include operators such as /, -, ^ as per their arithmetic associativity and precedence.	41
	2.25.2	Code	41
	2.25.3	Output	44
	2.26	Experiment - 26	45 - 46
	2.26.1	Question: Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method) for the following grammars: a) $S \rightarrow cAd, A \rightarrow ab/a$ b) $S \rightarrow cAd, A \rightarrow a/ab$	45
	2.26.2	Code	45
	2.26.3	Output	46

	2.27	Experiment - 27	47 - 48
	2.27.1	Question: Use YACC to generate 3-Address code for a given expression.	47
	2.27.2	Code	47
	2.27.3	Output	48
	2.28	Experiment - 28	49 - 50
	2.28.1	Question: Write a program to design LALR parsing using YACC.	49
	2.28.2	Code	49
	2.28.3	Output	50
	2.29	Experiment - 29	51 - 59
	2.29.1	Question: Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method). a) $S \rightarrow aaSaa \mid aa$ b) $S \rightarrow aaaSaaa \mid aa$ c) $S \rightarrow aaaaSaaaa \mid aa$ d) $S \rightarrow aaaSaaa \mid aSa \mid aa$	51
	2.29.2	Code	51
	2.29.3	Output	59

1. Course Outcomes

CO1: Apply the fundamental concepts for the various phases of compiler design.

CO2: Analyse the syntax and semantic concepts of a compiler.

CO3: Design various types of parsers and Address code generation.

CO4: Implement compiler principles, methodologies using lex, yacc tools.

2. Experiments

2.1 Experiment - 1

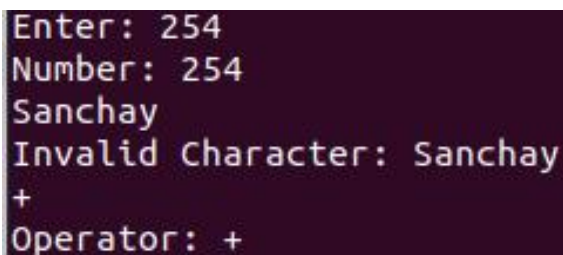
2.1.1 Question:

LEX program to check whether the entered user input is a number, operator, or invalid character.

2.1.2 Code:

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
[0-9]+ {printf("Number: %s\n", yytext);}
[+-] {printf("Operator: %s\n", yytext);}
[\t\n] {/*ignore Whitespaces and newline*/}
[a-zA-Z]* {printf("Invalid Character: %s\n", yytext);}
%%
int main()
{
printf("Enter: ");
yylex();
return 0;
}
```

2.1.3 Output:



```
Enter: 254
Number: 254
Sanchay
Invalid Character: Sanchay
+
Operator: +
```

2.2 Experiment - 2

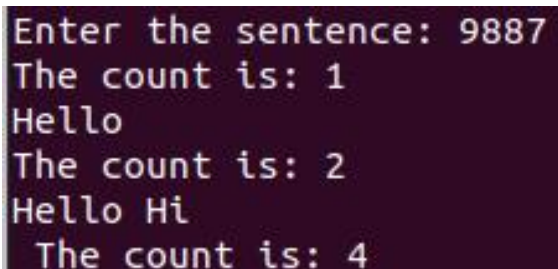
2.2.1 Question:

LEX program to perform word count on the entered user input.

2.2.2 Code:

```
%{
#include<stdio.h>
int c=0;
%}
%%
[a-zA-Z0-9]+ {c++;}
\n {printf("The count is: %d\n", c);}
%%
int yywrap()
{
}
int main()
{
printf("Enter the sentence: ");
yylex();
return 0;
}
```

2.2.3 Output:



```
Enter the sentence: 9887
The count is: 1
Hello
The count is: 2
Hello Hi
The count is: 4
```

2.3 Experiment - 3

2.3.1 Question:

LEX program to perform vowel count and consonant count.

2.3.2 Code:

```
%{
#include<stdio.h>
int vow_count=0;
int const_count=0;
%}
%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
\n {printf("Vowel Count: %d, Consonants Count: %d", vow_count, const_count);}
%%
int yywrap()
{
}
int main()
{
printf("Enter the string of Vowels and Consonants: ");
yylex();
return 0;
}
```

2.3.3 Output:

```
Enter the string of Vowels and Consonants: Hello
Vowel Count: 2, Consonants Count: 3^C
```

2.4 Experiment - 4

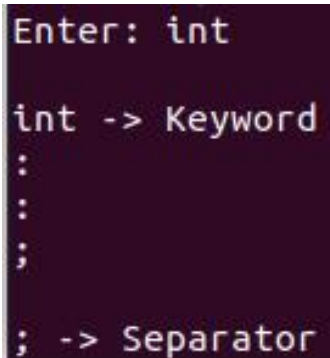
2.4.1 Question:

LEX program to identify whether the entered user input is a keyword, separator or identifier.

2.4.2 Code:

```
%option noyywrap
%{
#include<stdio.h>
%}
%%
int|char|float {printf("\n%s -> Keyword", yytext);}
,|; {printf("\n%s -> Separator", yytext);}
[a-zA-Z0-9]* {printf("\n%s -> Identifier", yytext);}
%%
int wrap()
{
}
int main()
{
printf("Enter: ");
yylex();
return 0;
}
```

2.4.3 Output:



```
Enter: int
int -> Keyword
:
:
;
; -> Separator
```

2.5 Experiment - 5

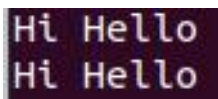
2.5.1 Question:

LEX program to print the same input that the user entered again using ECHO.

2.5.2 Code:

```
%%  
. ECHO;  
%%  
  
int yywrap(void)  
{  
}  
int main(void)  
{  
    yylex();  
    return 0;  
}
```

2.5.3 Output:

A screenshot of a terminal window with a dark background. It shows the output of the LEX program. The text "Hi Hello" is printed on the first line, and "Hi Hello" is printed on the second line, indicating that the input was echoed back to the user.

```
Hi Hello  
Hi Hello
```

2.6 Experiment - 6

2.6.1 Question:

LEX program to check whether input is digit or not.

2.6.2 Code:

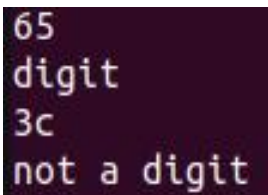
```
%{
#include<stdio.h>
#include<stdlib.h>
%}

%%
^[0-9]* {printf("digit");}
^[^0-9][0-9]*[a-zA-Z] {printf("not a digit");}
.;
%%

int yywrap()
{
}

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

2.6.3 Output:



```
65
digit
3c
not a digit
```


2.7 Experiment - 7

2.7.1 Question:

LEX program to check whether the given number is even or odd.

2.7.2 Code:

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

%%
[0-9]+ {i=atoi(yytext);
if(i%2==0)
printf("Even");
else
printf("Odd");}
%%

int yywrap()
{
}

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

2.7.3 Output:



```
3
Odd
2
Even
```

2.8 Experiment - 8

2.8.1 Question:

LEX program to check whether a number is prime or not.

2.8.2 Code:

```
%{
#include<stdio.h>
#include<stdlib.h>
int flag,c,j;
%}

%%
[0-9]+ {c=atoi(yytext);
        if(c==2)
        {
            printf("\n Prime number");
        }
        else if(c==0 || c==1)
        {
            printf("\n Not a Prime number");
        }
        else
        {
            for(j=2;j<c;j++)
            {
                if(c%j==0)
                flag=1;
            }
            if(flag==1)
            printf("\n Not a prime number");
            else if(flag==0)
            printf("\n Prime number");
        }
    }
%%

int yywrap()
{}

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

2.8.3 Output:

```
3
  Prime number
4
  Not a prime number
```

2.9 Experiment - 9

2.9.1 Question:

LEX program to print invalid string if a alpha numeric string is entered as an input.

2.9.2 Code:

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

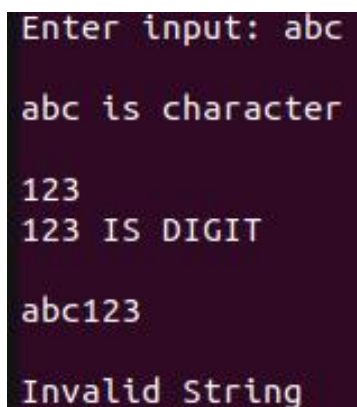
alpha [a-zA-Z0-9]*

%%
[0-9]* {printf("%s IS DIGIT\n",yytext);}
[a-zA-Z]* {printf("\n%s is character\n",yytext);}
{alpha} {printf("\nInvalid String\n");}
%%

int yywrap()
{
}

int main()
{
printf("Enter input: ");
yylex();
return 0;
}
```

2.9.3 Output:



```
Enter input: abc
abc is character
123
123 IS DIGIT
abc123
Invalid String
```

2.10 Experiment - 10

2.10.1 Question:

LEX program to recognize identifiers, keyword (int and float), anything else as invalid tokens.

2.10.2 Code:

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

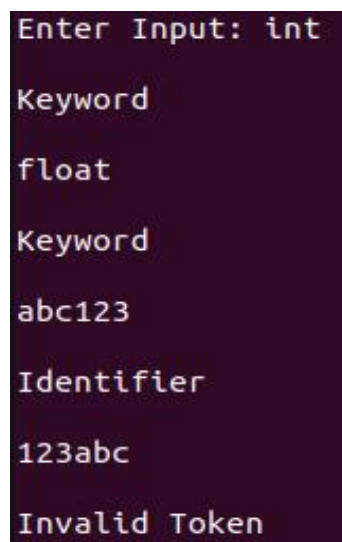
alpha[a-zA-Z]
digit[0-9]

%%
(float|int) {printf("\nKeyword\n");}
{alpha}({digit}|{alpha})* {printf("\nIdentifier\n");}
{digit}({digit}|{alpha})* {printf("\nInvalid Token\n");}
%%

int yywrap()
{
}

int main()
{
printf("Enter Input: ");
yylex();
return 0;
}
```

2.10.3 Output:



```
Enter Input: int
Keyword
float
Keyword
abc123
Identifier
123abc
Invalid Token
```

2.11 Experiment - 11

2.11.1 Question:

LEX program to recognize identifiers, keyword (int and float), anything else as invalid tokens. Read these from a text file.

2.11.2 Code:

```
%{
#include<stdio.h>
char fname[25];
%}

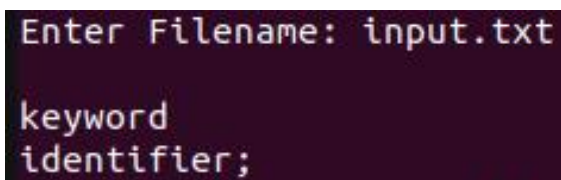
alpha[a-zA-Z]
digit[0-9]

%%
(float|int) {printf("\nkeyword");}
{alpha}({digit}|{alpha})* {printf("\nidentifier");}
{digit}({digit}|{alpha})* {printf("\ninvalid token");}
%%

int yywrap()
{
}

int main()
{
printf("Enter Filename: ");
scanf("%s",fname);
yyin=fopen(fname,"r");
yylex();
return 0;
fclose(yyin);
}
```

2.11.3 Output:



```
Enter Filename: input.txt
keyword
identifier;
```

2.12 Experiment - 12

2.12.1 Question:

- a) LEX program to identify alphabets as characters and numbers as digits.

Expected Output:

Enter input: sum33

Sum is a stream of characters

33 is a digit

- b) Modify the above program to get the following output:

Enter input: sum33

s is a character

u is a character

m is a character

3 is a digit

3 is a digit

2.12.2 Code:

a)

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

```
%%
```

```
[0-9]+ {printf("%s is digit\n", yytext);}  
[a-zA-Z]+ {printf("%s is stream of characters\n",yytext);}  
. { /* Ignore any other characters */ }
```

```
%%
```

```
int yywrap()  
{  
}
```

```
int main() {  
    printf("Enter input: ");  
    yylex();  
    return 0;  
}
```

b)

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

```
%%
```

```
[0-9]    {printf("%s is digit\n", yytext);}
[a-zA-Z] {printf("%s is character\n",yytext);}
.        { /* Ignore any other characters */ }
```

```
%%
```

```
int yywrap()
```

```
{
}
```

```
int main() {
    printf("Enter input: ");
    yylex();
    return 0;
}
```

2.12.3 Output:

```
Enter input: Hello21
Hello is stream of characters
21 is digit
```

```
Enter input: Hello21
H is character
e is character
l is character
l is character
o is character
2 is digit
1 is digit
```


2.13 Experiment - 13

2.13.1 Question:

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 sentence else it is simple sentence.

2.13.2 Code:

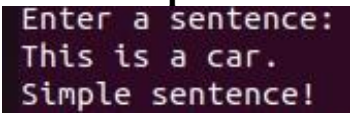
```
%{
    #include<stdio.h>
    int flag=0;
}%

%%
and |
or |
but |
because |
if |
then |
nevertheless { flag=1; }
. ;
\n { return 0; }
%%

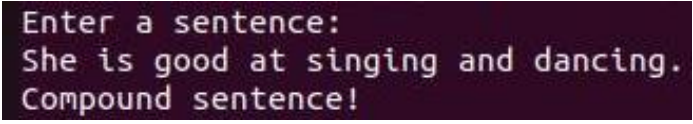
int main()
{
    printf("Enter the sentence:\n");
    yylex();
    if(flag==0)
        printf("Simple sentence\n");
    else
        printf("compound sentence\n");
}

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

2.13.3 Output:



```
Enter a sentence:
This is a car.
Simple sentence!
```



```
Enter a sentence:
She is good at singing and dancing.
Compound sentence!
```

2.14 Experiment - 14

2.14.1 Question:

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).

If sentence starts with the article, appropriate message should be printed. If sentence does not start with the article, appropriate message should be printed.

2.14.2 Code:

```
%option noyywrap
%{
    #include<stdio.h>
    int a=0;
}%

%%
[A|a|AN|An|THE|The] {a=1;}
[a-zA-Z]* {}
\n {return 0;}
%%

void main()
{
    printf("Enter a sentence \n");
    a=0;
    yylex();
    if(a==1)
    {
        printf("The sentence STARTS with an ARTICLE");
    }
    else
    {
        printf("The sentence DOES NOT start with an Article");
    }
}
```

2.14.3 Output:

```
Enter a sentence:
This is a good idea.
Does not start with an article!
```

```
Enter a sentence:
The sun rises in the east.
Starts with an article!
```

2.15 Experiment - 15

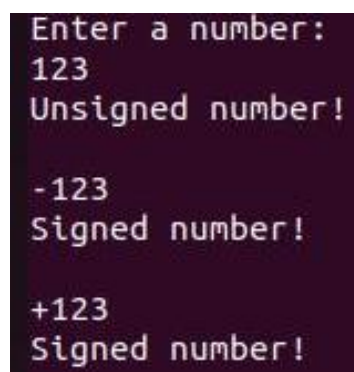
2.15.1 Question:

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

2.15.2 Code:

```
%option noyywrap
%{
    #include<stdio.h>
    int a=0;
}%
%%
[+|-] {a=1;}
[0-9]* {}
\n {return 0;}
%%
void main()
{
    printf("Enter a Number: ");
    a=0;
    yylex();
    if(a==1)
    {
        printf("Signed Number!\n");
    }
    else
    {
        printf("Unsigned Number!\n");
    }
}
```

2.15.3 Output:



```
Enter a number:
123
Unsigned number!

-123
Signed number!

+123
Signed number!
```

2.16 Experiment - 16

2.16.1 Question:

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

2.16.2 Code:

```
%option noyywrap
%{
    #include<stdio.h>
    int a=0;
}%

%%
[a-zA-Z]+[?|.!] {a=1;}
\n {return 0;}
%%

void main()
{
    printf("Enter a sentence: ");
    a=0;
    yylex();
    if(a==1)
    {
        printf("Ends with a punctuation!\n");
    }
    else
    {
        printf("Does not end with a punctuation!\n");
    }
}
```

2.16.3 Output:

```
Enter a sentence:
Is this yours?
Ends with a punctuation!
```

```
Enter a sentence:
You are good
Does not end with punctuation!
```

2.17 Experiment - 17

2.17.1 Question:

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.

2.17.2 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);
}
```

2.17.3 Output:

```
Enter a sentence:
//This is a comment.
No of comment lines are: 1
/*This is multi*/ //This is single.
No of comment lines are: 2
There are no comments.
There are no comments.No of comment lines are: 0
```

2.18 Experiment - 18

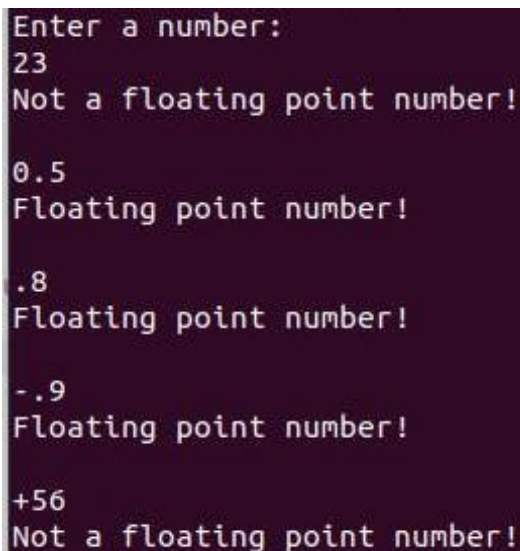
2.18.1 Question:

Write a program in LEX to recognize Floating Point Numbers.

2.18.2 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();
}
```

2.18.3 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!
```

2.19 Experiment - 19

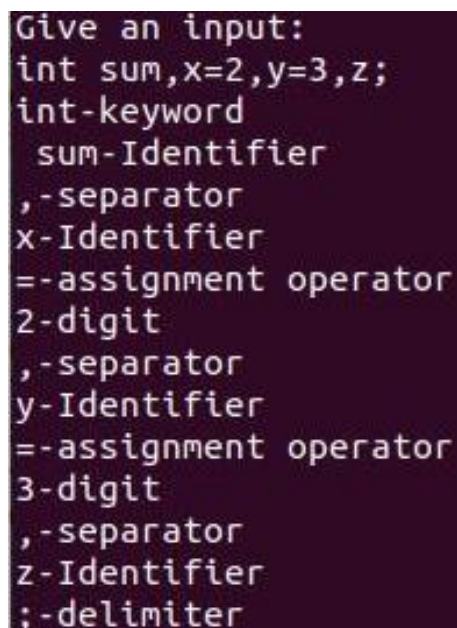
2.19.1 Question:

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

2.19.2 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;
}
```

2.19.3 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
```

2.20 Experiment - 20

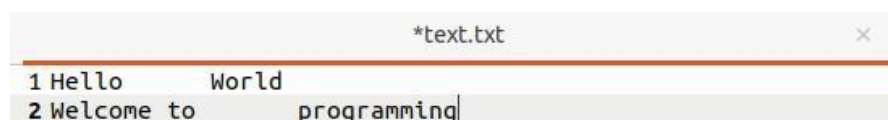
2.20.1 Question:

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

2.20.2 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;
}
```

2.20.3 Output:



```
1 Hello      World
2 Welcome to  programming|
```

Printed!



```
1 Hello World
2 Welcome to programming
```


2.21 Experiment - 21

2.21.1 Question:

Write a LEX program to recognize the following tokens over the digits [0-9]:

- a) The set of all string ending in 00.
- b) The set of all strings with three consecutive 222's.
- c) The set of all string such that every block of five consecutive symbols contains at least two 5's.
- d) The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.
- e) The set of all strings such that the 10th symbol from the right end is 1.
- f) The set of all four digits numbers whose sum is 9.
- g) The set of all four digital numbers, whose individual digits are in ascending order from left to right.

2.21.2 Code:

```
a)
%{
#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;
}
```

```
b)
%{
#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;
}

```

c)

```

%{
#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;
}

```

```

d)
%{
#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;
}

```

```

e)
%{
#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;
}

```

```

f)
%{
#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;
}

```

```

g)
%{
#include<stdio.h>
int c,i,flag=1;
%}
%%
[0-9][0-9][0-9][0-9] {for(i=0;i<yy leng-1;i++)
{
if(yytext[i]>=yytext[i+1])
{
flag=0;
break;
}
}
}
\n {return 0;}
%%
void main()
{
printf("Enter a string:\n");
yylex();

```

```

if(flag==1)
printf("The digits are in ascending order.\n");
else
printf("The digits are not in ascending order.\n");
}
int yywrap()
{
return 1;
}

```

2.21.3 Output:

a)

```

Enter a string:
12300
Ends with 0.

```

```

Enter a string:
145
Does not end with 0.

```

b)

```

Enter a string:
2322
Does not have 3 consecutive 2's.

```

```

Enter a string:
322221
Has 3 consecutive 2's.

```

c)

```

Enter a string:
1525558566
yytext:15255,flag(1 if no of 5 is atleast 2):1
yytext:58566,flag(1 if no of 5 is atleast 2):1
Valid string.

```

```

Enter a string:
12345455
yytext:12345,flag(1 if no of 5 is atleast 2):0
Not a valid string!

```

d)

```

Enter a string:
1010
Decimal representation:10
Congruent to modulo 5.

```

```

Enter a string:
111
Decimal representation:7
Not congruent to modulo 5.

```

```
Enter a string:  
123  
Not a binary number.
```

e)

```
Enter a string:  
11234345236  
10th symbol from right is 1.
```

```
Enter a string:  
23123456123  
10th symbol from right is not 1.
```

f)

```
Enter a string:  
6300  
The sum of digits is 9.
```

```
Enter a string:  
3331  
The sum of digits is not 9.
```

g)

```
Enter a string:  
1235  
The digits are in ascending order.
```

```
Enter a string:  
1243  
The digits are not in ascending order.
```

2.22 Experiment - 22

2.22.1 Question:

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

2.22.2 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;
}
```


2.22.3 Output:

```
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: }
```

2.23 Experiment - 23

2.23.1 Question:

- a) Design a suitable grammar for evaluation of arithmetic expression having + and – operators.
+ has least priority and it is left associative
- has higher priority and is right associative
- b) Design a suitable grammar for evaluation of arithmetic expression having + , – , * , / , %, ^ operators.
^ having highest priority and right associative
% having second highest priority and left associative
* , / have third highest priority and left associative
+ , - having least priority and left associative

2.23.2 Code:

a)

arithmetic.l

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

arithmetic.y

```
%{  
#include <stdio.h>  
%}  
  
%token NUM  
%left '+'  
%right '-'  
  
%%  
expr:e {printf("Valid expression\n"); printf("Result: %d\n", $$); return 0;}  
e:e+e {$$=$1+$3;}  
| e'-e {$$=$1-$3;}  
| NUM {$$=$1;}  
;
```

```
%%
```

```
int main()
{
    printf("\nEnter an arithmetic expression\n");
    yyparse();
    return 0;
}
```

```
int yyerror()
{
    printf("\nInvalid expression\n");
    return 0;
}
```

b)
arithmetic_modified.l

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
```

```
%%
[0-9]+ {
    yylval=atoi(yytext);
    return NUMBER;
}
```

```
[\t] ;
[\n] return 0;
. return yytext[0];
%%
```

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

arithmetic_modified.y

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

```
%token NUMBER
```

```
%right '^'
```

```
%left '%'
%left '*' '/'
%left '+' '-'
%left '(' ')'
```

```
%%
```

```
ArithmeticExpression: E{
    printf("\nResult=%d\n", $$);
    return 0;
}
```

```
E: E '^' E { $$ = pow($1, $3); }
```

```
| E '%' E { $$ = $1 % $3; }
```

```
| E '*' E { $$ = $1 * $3; }
```

```
| E '/' E { $$ = $1 / $3; }
```

```
| E '+' E { $$ = $1 + $3; }
```

```
| E '-' E { $$ = $1 - $3; }
```

```
| '(' E ')' { $$ = $2; }
```

```
| NUMBER { $$ = $1; }
```

```
;
```

```
%%
```

```
void main()
```

```
{
```

```
    printf("\nEnter Any Arithmetic Expression which can have operations Addition,
Subtraction, Multiplication, Divison, Modulus, Exponentiation, and Round brackets:\n");
```

```
    yyparse();
```

```
    if(flag==0)
```

```
        printf("\nEnter arithmetic expression is Valid\n\n");
```

```
}
```

```
void yyerror(const char *s)
```

```
{
```

```
    printf("\nEnter arithmetic expression is Invalid\n\n");
```

```
    flag=1;
```

```
}
```

2.23.3 Output:

a)

```
Enter an arithmetic expression
5+6-3-6
Valid expression
Result : 14
```

b)

```
Enter an arithmetic expression:
2+3*4
Valid expression!
Result:14
Enter an arithmetic expression:
2++3-
Invalid expression!
```

2.24 Experiment - 24

2.24.1 Question:

- a) Use YACC to generate Syntax tree for a given expression.
- b) Update the above program based on the following priority and associativity:
 - + , - having least priority and left associative
 - * , / have second highest priority and left associative
 - ^ having next highest priority and right associative
 - () having highest priority

2.24.2 Code:

a)

syntax_tree.l

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

syntax_tree.y

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

```
struct tree_node
{
    char val[20];
    int lc;
    int rc;
};
```

```
int ind;
struct tree_node syn_tree[100];
```

```

extern int yylex(void);
int yyerror(const char *s);
void my_print_tree(int cur_ind);
int mknode(int lc, int rc, const 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 ')' { $$ = $2; }
  | digit { char buf[20]; sprintf(buf, "%d", yylval); $$ = mknode(-1, -1, buf); }
;

%%

int main()
{
    ind = 0;
    printf("Enter an expression\n");
    yyparse();
    return 0;
}

int yyerror(const char *s)
{
    printf("NITW Error: %s\n", s);
}

int mknode(int lc, int rc, const char* val)
{
    if (strlen(val) < sizeof(syn_tree[ind].val))
    {
        strcpy(syn_tree[ind].val, val);
    }
}

```

```

    }
    else
    {
        fprintf(stderr, "Error: Value too large for tree node\n");
        exit(EXIT_FAILURE);
    }

    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\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);
}

```

b)
`syntax_tree_modified.l`

```

%{
#include "y.tab.h"
extern int yylval;
}%

%%

[0-9]+ { yylval=atoi(yytext); return digit;}
[t] ;
[n] return 0;
. return yytext[0];
%%

int yywrap()
{}

```

syntax_tree_modified.y

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

extern int yylex();
extern void yyerror(const char *s);

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, const char *val);
%}
%token digit
%right '^'
%left '*' '/'
%left '+' '-'
%%
S: E { my_print_tree($1); }
    ;

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

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

F: '(' E ')' { $$ = $2; }
    | digit { char buf[10]; sprintf(buf, "%d", yylval); $$ = mknode(-1, -1, buf); }
    | F '^' F { $$ = mknode($1, $3, "^"); }
    ;

%%
int main()
```



```

{
    ind = 0;
    printf("Enter an expression\n");
    yyparse();
    return 0;
}

void yyerror(const char *s)
{
    printf("NITW Error: %s\n", s);
    exit(EXIT_FAILURE);
}

int mknode(int lc, int rc, const char *val)
{
    strcpy(syn_tree[ind].val, val);
    syn_tree[ind].lc = lc;
    syn_tree[ind].rc = rc;
    ind++;
    return ind - 1;
}

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);
}

```

2.24.3 Output:

a)

```
Enter an expression
2+3*5
Operator Node -> Index: 4, Value: +, Left Child Index: 0, Right Child Index: 3
Digit Node -> Index: 0, Value: 2
Operator Node -> Index: 3, Value: *, Left Child Index: 1, Right Child Index: 2
Digit Node -> Index: 1, Value: 3
Digit Node -> Index: 2, Value: 5
```

b)

```
Enter an expression
3+4*(2-1)^2
Operator Node -> Index : 8, Value : +, Left Child Index : 0, Right Child Index : 7
Digit Node -> Index : 0, Value : 3
Operator Node -> Index : 7, Value : *, Left Child Index : 1, Right Child Index : 6
Digit Node -> Index : 1, Value : 4
Operator Node -> Index : 6, Value : ^, Left Child Index : 4, Right Child Index : 5
Operator Node -> Index : 4, Value : -, Left Child Index : 2, Right Child Index : 3
Digit Node -> Index : 2, Value : 2
Digit Node -> Index : 3, Value : 1
Digit Node -> Index : 5, Value : 2
```

2.25 Experiment - 25

2.25.1 Question:

- Use YACC to convert: Infix expression to Postfix expression.
- Modify the above program so as to include operators such as /, -, ^ as per their arithmetic associativity and precedence

2.25.2 Code:

a)

Infix.l

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

Infix.y

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

```
int yylex(void); // Declaration of yylex
```

```
// Declaration of yyerror
int yyerror(const char *s);
}%
```

```
%token digit
```

```
%%
S: E {printf("\n\n");}
;
E: E '+' T { printf("+");}
| T
;
T: T '*' F { printf("*");}
| F
```

```

;
F: '(' E ')'
| digit {printf("%d", $1);}
;

```

```
%%
```

```

int main()
{
    printf("Enter infix expression: ");
    yyparse();
    return 0;
}

```

```

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

```

b)

Infix_modified.l

```

%{
#include "y.tab.h"
extern int yylval;
%}

```

```
%%
```

```

[0-9]+ { yylval=atoi(yytext); return digit; }
[\t] ;
[\n] return 0;
. return yytext[0];

```

```
%%
```

```

int yywrap() {
    // Add implementation if needed
}

```

Infix_modified.y

```

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

```

```

extern int yylex(); // Declare the yylex function

```

```
extern void yyerror(char *s); // Declare the yyerror function
```

```
%}
```

```
%token digit
```

```
%left '+' '-'
```

```
%left '*' '/'
```

```
%right '^'
```

```
%%
```

```
S: E { printf("\n\n"); }
```

```
;
```

```
E: E '+' T { printf("+"); }
```

```
  | E '-' T { printf("-"); }
```

```
  | T
```

```
;
```

```
T: T '*' F { printf("*"); }
```

```
  | T '/' F { printf("/"); }
```

```
  | F
```

```
;
```

```
F: F '^' G { printf("^"); }
```

```
  | G
```

```
;
```

```
G: '(' E ')'
```

```
  | digit { printf("%d", $1); }
```

```
;
```

```
%%
```

```
int main() {
```

```
    printf("Enter infix expression: ");
```

```
    yyparse();
```

```
}
```

```
void yyerror(char *s) {
```

```
    printf("Error: %s\n", s);
```

```
    // You can add more error handling if needed
```

```
}
```

2.25.3 Output:

a)

```
Enter infix expression: 3*4+5*2
34*52*+
```

b)

```
Enter infix expression: 3-5*4+1/3
354*-13/+
```

2.26 Experiment - 26

2.26.1 Question:

Write a program to implement:

Recursive Descent Parsing with back tracking (Brute Force Method) for the following grammars:

a) $S \rightarrow cAd$, $A \rightarrow ab / a$

b) $S \rightarrow cAd$, $A \rightarrow a / ab$

2.26.2 Code:

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

```
int i = 0;
```

```
bool A(char *s)
{
    if(s[i] != 'a')
        return false;
    if(s[i] == 'a')
        i++;
    if(i < strlen(s) && s[i] == 'b')
        i++;
    return true;
}
```

```
bool S(char *s)
{
    if(strlen(s) == 0)
        return false;
    if(s[i] == 'c')
    {
        i++;
        if(!A(s))
            return false;
        if(i == strlen(s))
            return false;
        if(s[i] == 'd')
            i++;
        if(i == strlen(s))
            return true;
    }
    return false;
}
```

```
int main()
```

```
{  
    char s[100];  
    printf("Enter the string: ");  
    scanf("%99s", s);  
    if(S(s))  
        printf("Input matched\n");  
    else  
        printf("Input did not match\n");  
}
```

2.26.3 Output:

```
Enter the string: cabd  
Input matched
```

```
Enter the string: cadb  
Input did not match
```


2.27 Experiment - 27

2.27.1 Question:

Use YACC to generate 3-Address code for a given expression.

2.27.2 Code:

3address.l

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

3address.y

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>
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;
}
yyerror()
{
printf("error");
}

```

2.27.3 Output:

```

Enter an expression :
a=2+3*6
t0 = 2;
t1 = 3;
t2 = 6;
t3 = t1 * t2;
t4 = t0 + t3;
a=t4

```

```

Enter an expression :
a=4*5-2
t0 = 4;
t1 = 5;
t2 = t0 * t1;
t3 = 2;
t4 = t2 - t3;
a=t4

```

2.28 Experiment - 28

2.28.1 Question:

Write a program to design LALR parsing using YACC.

2.28.2 Code:

Lalr.l

```
%{
#include "y.tab.h"
extern int yylval;
}%
%%
//If the token is an Integer number,then return it's value. [0-9]+ {yylval=atoi(yytext); return
digit;}
//If the token is space or tab,then just ignore it.
[\t] ;
//If the token is new line,return 0.
[\n] return 0;
//For any other token, return the first character read since the last match.
. return yytext[0];
%%
```

Lalr.y

```
%{
/* Definition section */
#include <ctype.h>
#include<stdio.h>
#include<stdlib.h>
}%
%token digit
/* Rule Section */
%%
/*After Evaluating the expression E,S prints Reached*/ S: E {printf("Reached\n\n");}
;
/*The expression parser uses three different symbols T,F and P, to set the
precedence and associativity of operators*/
E: E '+' T
| E '-' T
| T
;
T: T '*' P
| T '/' P
| P
;
P: F '^' P
| F
;
```

```
F: '(' E ')'
| digit
;
%%
```

```
//driver code
int main()
{
    printf("Enter infix expression: ");
    yyparse();
}
yyerror()
{
    printf("NITW Error");
}
```

2.28.3 Output:

A terminal window with a dark background and light-colored text. The first line shows the prompt "Enter infix expression:" followed by the input "2+3*4". The second line shows the output "Reached".

```
Enter infix expression: 2+3*4
Reached
```

2.29 Experiment - 29

2.29.1 Question:

Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method).

- a) $S \rightarrow aaSaa \mid aa$
- b) $S \rightarrow aaaSaaa \mid aa$
- c) $S \rightarrow aaaaSaaaa \mid aa$
- d) $S \rightarrow aaaSaaa \mid aSa \mid aa$

2.29.2 Code:

```
a)
/* S->aaSaa | aa */
#include<bits/stdc++.h>
using namespace std;
int curr;
//??
int S(char b[],int l)
{
//match with aa
char prod[20];
int isave=curr;
strcpy(prod,"aaSaa");
if(curr<l && b[curr]=='a')
{
curr++;
if(curr<l && b[curr]=='a')
{
curr++;
//recursive call to match S
if(S(b,l))
{
if(curr<l && b[curr]=='a')
{
curr++;
if(curr<l && b[curr]=='a')
{
curr++;
return 1;
}
}
}
}
}
}
```

```

//match with aa
strcpy(prod,"aa");
curr=isave;
if(curr<l && b[curr]=='a')
1
{
curr++;
if(curr<l && b[curr]=='a')
{
curr++;
return 1;
}
}
return 0;
}
int main()
{
curr=0;
char a[500];
cout<<"Enter the string : ";
cin.getline(a,500,'\n');
int l=strlen(a);
cout<<"length = "<<l<<endl;
if(S(a,l) && curr==l)
{
cout<<"Accepted\n";
}
else
{
cout<<"Not Accepted\n";
}
return 0;
}

```

```

#include<bits/stdc++.h>
using namespace std;
int i;
//tries all possible centres recursively and try to match the string
int S(char b[],int l)
{
int isave=i;
//match with aa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')

```

```

{
i++;
//match with S recursively
if(S(b,l))
{
//match with aa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
return 1;
}
}
}
}
}
i=isave;
//match with middle aa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
return 1;
}
}
return 0;
}
int main()
{
i=0;
char a[500];
memset(a,'\0',500);
for(int j=0;j<400;j++)
{
a[j]='a';
i=0;
if(S(a,j+1) && i==j+1)
{
cout<<j+1<<" ";
}
}
}
return 0;

```

```
}
```

b)

```
#include<bits/stdc++.h>
using namespace std;
int i;
//??
//checks for grammer S->aaaSaaa | aa
//tries all possible centres recursively and try to match the string
int S(char b[],int l)
{
    int isave=i;
    //match with aaa
    if(i<l && b[i]=='a')
    {
        i++;
        if(i<l && b[i]=='a')
        {
            i++;
            if(i<l && b[i]=='a')
            {
                i++;
                //match with S recursively
                if(S(b,l))
                {
                    //match with aaa
                    if(i<l && b[i]=='a')
                    {
                        i++;
                        if(i<l && b[i]=='a')
                        {
                            i++;
                            if(i<l && b[i]=='a')
                            {
                                i++;
                                return 1;
                            }
                        }
                    }
                }
            }
        }
    }
    i=isave;
    //match with middle aa
    if(i<l && b[i]=='a')
```



```

{
i++;
if(i<l && b[i]=='a')
{
i++;
return 1;
}
}
return 0;
}
int main()
{
i=0;
char a[500];
memset(a,'\0',500);
for(int j=0;j<400;j++)
{
a[j]='a';
i=0;
if(S(a,j+1) && i==j+1)
{
cout<<j+1<<" ";
}
}
return 0;
}

```

c)

```

#include<bits/stdc++.h>
using namespace std;
int i;
//??
//checks for grammer S->aaaaSaaaa | aa
//tries all possible centres recursively and try to match the string
int S(char b[],int l)
{
int isave=i;
//match with aaaa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{

```

```

i++;
if(i<l && b[i]=='a')
{
i++;
//match with S recursively
if(S(b,l))
{
//match with aaaa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
if(i<l &&
b[i]=='a')
{
i++;
return 1;
}
}
}
}
}
}
}
}
}
}
}
}
}
}
i=isave;
//match with middle aa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
return 1;
}
}
return 0;
}
int main()
{

```

```

i=0;
char a[500];
memset(a,'\0',500);
for(int j=0;j<400;j++)
{
a[j]='a';
i=0;
if(S(a,j+1) && i==j+1)
{
cout<<j+1<<" ";
}
}
return 0;
}

```

d)

```

#include<bits/stdc++.h>
using namespace std;
int i;
//??
//checks for grammer S->aaaSaaa | aSa | aa
//tries all possible centres recursively and try to match the string
int S(char b[],int l)
{
int isave=i;
//match with aaa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
//match with S recursively
if(S(b,l))
{
//match with aaa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')

```

```

{
i++;
return 1;
}
}
}
}
}
}
}
}
i=isave;
//match with a
if(i<l && b[i]=='a')
{
i++;
//match with S recursively
10
if(S(b,l))
{
//match with a
if(i<l && b[i]=='a')
{
i++;
return 1;
}
}
}
i=isave;
//match with middle aa
if(i<l && b[i]=='a')
{
i++;
if(i<l && b[i]=='a')
{
i++;
return 1;
}
}
return 0;
}
int main()
{
i=0;
char a[500];
memset(a,'\0',500);
for(int j=0;j<400;j++)

```

```

{
a[j]='a';
i=0;
if(S(a,j+1) && i==j+1)
{
cout<<j+1<<" ";
}
}
return 0;
}

```

2.29.3 Output:

a)

```

Enter the string : aaaaaa
length = 6
Accepted

```

```

2 6 14 30 62 126 254

```

b)

```

2 8 20 44 92 188 380

```

c)

```

2 10 26 58 122 250

```

d)

```

2 4 8 16 32 64 128 256

```