

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT on COMPILER DESIGN

*Submitted by*

**Nishant kumar (1BM21CS117)**

*Under the Guidance of*

**Prof. SONIKA S**  
**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 **NISHANT KUMAR(1BM21CS117)** , who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfilment 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.

Prof. Sonika S  
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, Nishant kumar (1BM21CS117), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Sonika, 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      |
|----------|--|--------------|
| <b>1</b> |  | <b>6-8</b>   |
| 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            |
| <b>2</b> |  | <b>9-15</b>  |
| 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           |
| <b>3</b> |  | <b>16-23</b> |
| 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           |
| <b>4</b> |  | <b>24-36</b> |
| 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.                                      | 28-29        |
| 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. | 30-31        |
| 4.2.5    | The set of all strings such that the 10th symbol from the right end is 1.   | 32           |
| 4.2.6    | The set of all four digits numbers whose sum is 9.  | 33-34        |
| 4.2.7    | The set of all four digital numbers, whose individual digits are in ascending order from left to right.                                 | 35-36        |
| <b>5</b> |   | <b>37-38</b> |
| 5.1      | Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.          | 37-38        |
| <b>6</b> |   | <b>39-40</b> |
| 6.1      | Write a program to perform recursive descent parsing on the following grammar: $S \rightarrow cAd$<br>$A \rightarrow ab \mid a$         | 39-40        |
| <b>7</b> |   | <b>41-47</b> |
| 7.1      | Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.                      | 41-42        |
| 7.2      | Write a program in YACC to recognize strings of the form $\{(a^n)b, n \geq 5\}$ .   | 43-44        |
| 7.3      | Write a program in YACC to generate a syntax tree for a given arithmetic expression.  | 45-47        |
| <b>8</b> |   | <b>48-49</b> |
| 8.1      | Write a program in YACC to convert infix to postfix expression.   | 48-49        |
| <b>9</b> |   | <b>50-52</b> |
| 9.1      | Write a program in YACC to generate three address code for a given expression.  | 50-52        |

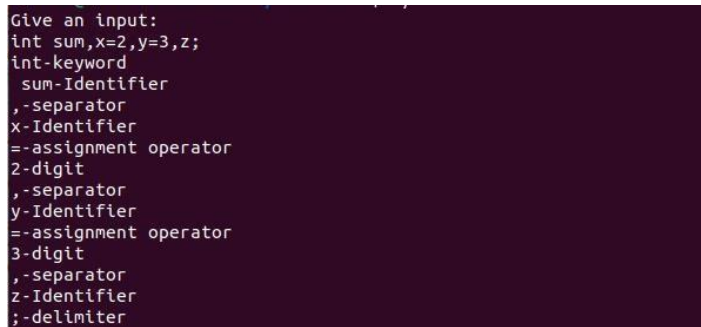
## Lab 1

### 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

```

Enter a sentence:
I was born in 2003.
No of characters and digits are 10 and 4
Hello123
No of characters and digits are 5 and 3

```

## 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");
```

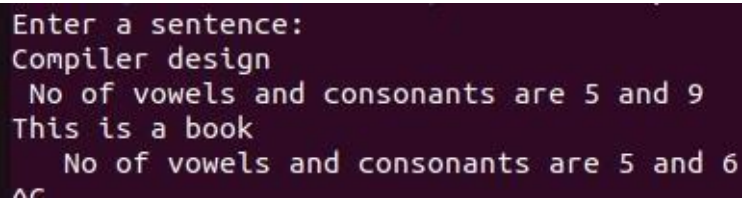
```
yylex(); } int yywrap()
```

```
{ return
```

```
1;
```

```
}
```

## Output



```
Enter a sentence:
Compiler design
No of vowels and consonants are 5 and 9
This is a book
No of vowels and consonants are 5 and 6
AC
```

## 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

```

Enter a sentence:
My name is Neha
No of words in the sentence are 4.
I will make things happen.
No of words in the sentence are 5.

```

## 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

```
Enter a string:
HelloWorld
Characters

1234
Digits
Hello123
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>

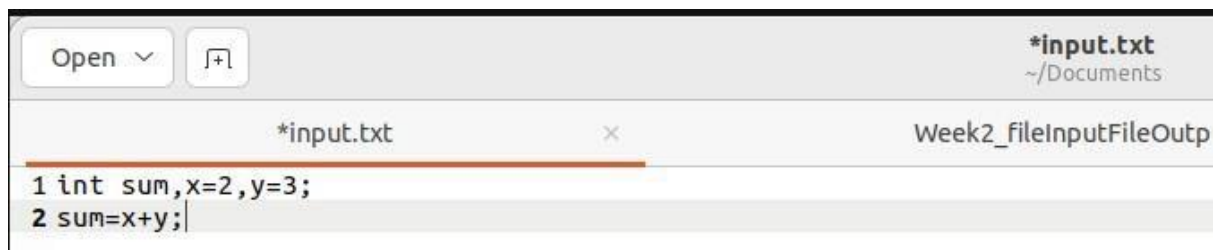
%} %%

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("input.txt","r");
yylex(); fclose(yyin);
} int
yywrap()
{ return
1;
```

}

## Output



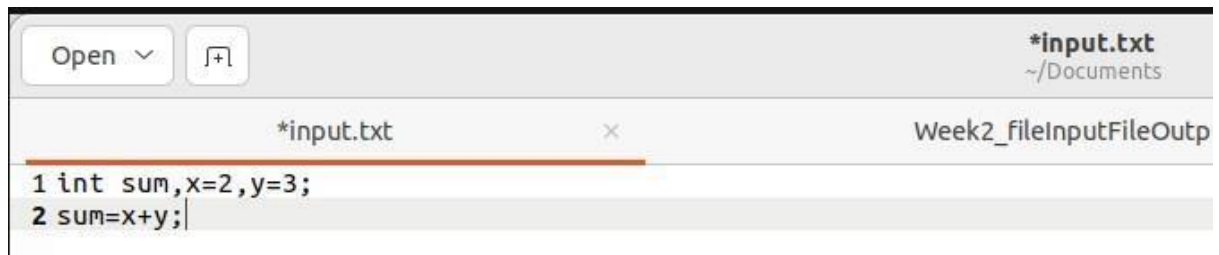
```
int is a keyword.  
  sum is an identifier.  
  , is a separator.  
x is an identifier.  
= is an assignment operator.  
2 is/are digit(s).  
  , is a separator.  
y is an identifier.  
= is an assignment operator.  
3 is/are digit(s).  
; is a delimiter.  
sum is an identifier.  
= is an assignment operator.  
x is an identifier.  
+ is a binary operator.  
y is an identifier.  
; is a delimiter.
```

## 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("input.txt","r");  
yyout=fopen("output.txt","w"); yylex();  
printf("Printed in output.txt\n");  
fclose(yyin); fclose(yyout);  
}  
int  
yywrap()  
{ return  
1;  
}
```

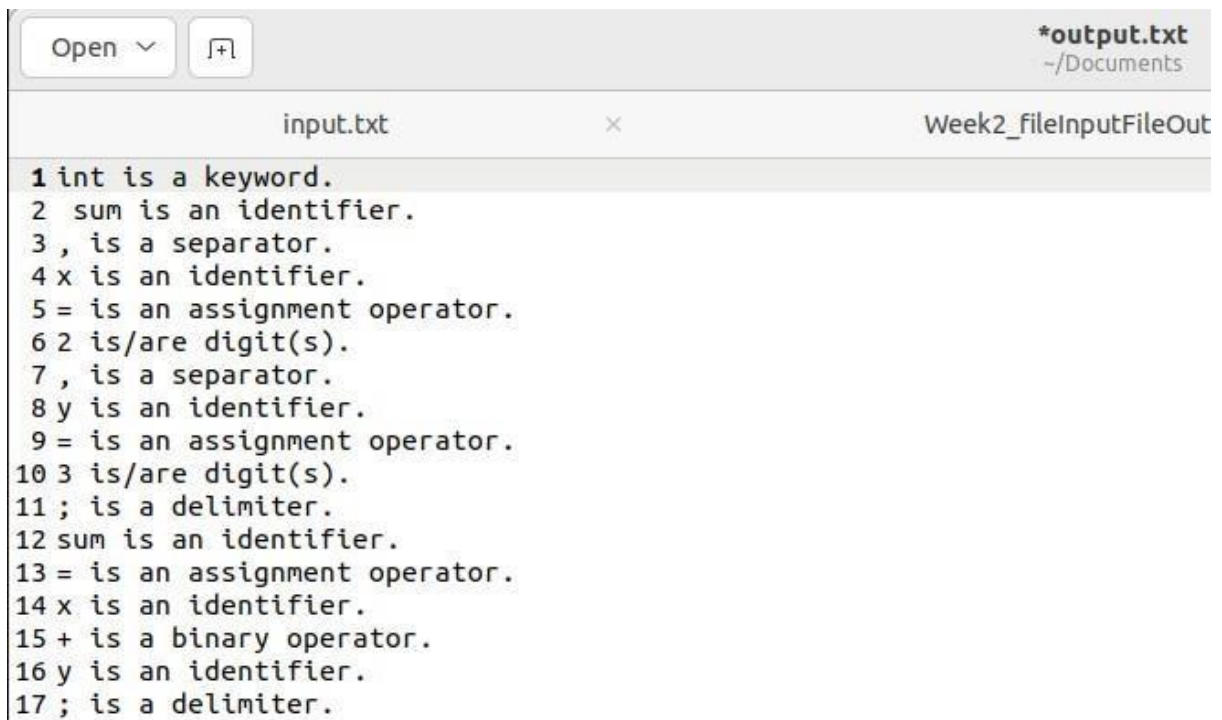
## Output



The screenshot shows a text editor window with a title bar containing "Open", a file icon, and the filename "\*input.txt" with the path "~/Documents". The editor has a tab labeled "input.txt" and a window title "Week2\_fileInputFileOutp". The content of the file is as follows:

```
1 int sum,x=2,y=3;
2 sum=x+y;
```

Printed in output.txt



The screenshot shows a text editor window with a title bar containing "Open", a file icon, and the filename "\*output.txt" with the path "~/Documents". The editor has a tab labeled "input.txt" and a window title "Week2\_fileInputFileOut". The content of the file is as follows:

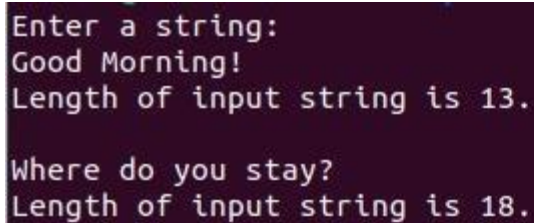
```
1 int is a keyword.
2 sum is an identifier.
3 , is a separator.
4 x is an identifier.
5 = is an assignment operator.
6 2 is/are digit(s).
7 , is a separator.
8 y is an identifier.
9 = is an assignment operator.
10 3 is/are digit(s).
11 ; is a delimiter.
12 sum is an identifier.
13 = is an assignment operator.
14 x is an identifier.
15 + is a binary operator.
16 y is an identifier.
17 ; is a delimiter.
```

## 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",yylen);}  
%%  
void main() {  
printf("Enter a string:\n");  
yylex(); } int yywrap() {  
return 1;  
}
```

### Output



```
Enter a string:  
Good Morning!  
Length of input string is 13.  
  
Where do you stay?  
Length of input string is 18.
```

## Lab 3

## 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

```
Printed!
```



## 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

```
Enter a string:  
12300  
Ends with 0.
```

```
Enter a string:  
145  
Does not end with 0.
```

### 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

```

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

```

### 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

```

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

```

#### 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=yytext-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

```

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

```

```

Enter a string:
101
Decimal representation:5
Congruent to modulo 5.

```

#### 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

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

#### 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<yylen;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; }
```

##### Output

```
Enter a string:
2340
The sum of digits is 9.
```

#### **4.2.7 The set of all four digital numbers, whose individual digits are in ascending order from left to right.**

##### **Code**

```
%{
#include<stdio.h> int
c,i,flag=1;
%}
%%
```



```

[0-9][0-9][0-9][0-9] {for(i=0;i<yyleng-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;
}

```

## Output

```

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

```

```

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

```

## 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

```
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

```
Enter a string:
cad$
Valid string!
```

```
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

```

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

```

## 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>=5.\nValid String!\n");}  
  
;  
  
S:S A  
  
|
```



```

;
%%

void main() {
printf("Enter a string!\n");
yyparse(); } int
yyerror(char *s) {
printf("Invalid String!\n");
return 0; }

```

### Output

```

Enter a string!
aaaaaaab
Parsed using the rule (a^n)b, n>=5.
Valid String!
ab
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

```

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

```

## 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

```

Enter an infix expression:
2+3*8/4^3-3
238*43^/+3-

```

## 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 yyval; extern
char iden[20];
%}
d [0-9]+ a
[a-zA-Z]+
%%
{d} { yyval=atoi(yytext); return digit; }
{a} { strcpy(iden,yytext); yyval=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");  
yyvsparse(); return 0; } int  
yyerror(char *s)  
{  
printf("Invalid expression!"); return  
0;  
}
```

## Output



```
Enter an expression:  
a=2*3/6-4  
t0 = 2;  
t1 = 3;  
t2 = t0 * t1;  
t3 = 6;  
t4 = t2 / t3;  
t5 = 4;  
t6 = t4 - t5;  
a=t6
```



