

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT

on

## COMPILER DESIGN

*Submitted by*

**Krishn Maloo (1BM21CS092)**

*Under the Guidance of  
Prof. Sunayana 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 **Krishn Maloo (1BM21CS092)**, 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. Sunayana 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, Krishn Maloo (1BM21CS092), 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. Sunayana S, 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

<b>Lab No</b>	<b>Title</b>
<b>1</b>	
1.1	Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.
1.2	Write a program in LEX to count the number of vowels and consonants in a string.
<b>2</b>	
2.1	Write a program in lex to count the number of words in a sentence.
2.2	Write a program in lex to demonstrate regular definition.
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.
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.
<b>3</b>	
3.1	Write a program in LEX to recognize Floating Point Numbers.
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.
3.3	Write a program to check if the input sentence ends with any of the following punctuation marks ( ?, fullstop , ! )
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).
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.
3.6	Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.
<b>4</b>	
4.1	Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.
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.
4.2.2	The set of all strings with three consecutive 222's.
4.2.3	The set of all string such that every block of five consecutive symbols contains at least two 5's.
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.
4.2.5	The set of all strings such that the 10th symbol from the right end is 1.
4.2.6	The set of all four digits numbers whose sum is 9.
4.2.7	The set of all four digital numbers, whose individual digits are in ascending order from left to right.
<b>5</b>	
5.1	Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.
<b>6</b>	
6.1	Write a program to perform recursive descent parsing on the following grammar: S->cAd A->ab   a
<b>7</b>	
7.1	Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.
7.2	Write a program in YACC to recognize strings of the form $\{(a^n)b, n \geq 5\}$ .
7.3	Write a program in YACC to generate a syntax tree for a given arithmetic expression.
<b>8</b>	
8.1	Write a program in YACC to convert infix to postfix expression.
<b>9</b>	
9.1	Write a program in YACC to generate three address code for a given expression.

# Lab 1

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

Code:

1. Identify datatype - int, char, float are variables.  
y. d  
#include <stdio.h>  
v. ;  
y. y.  
int float char & printf ("keyword"); ;  
[a-zA-Z]\* & printf ("Identifier"); ;  
, ; & printf ("separator"); ;  
y. y.  
int yywrap()  
d  
return 1;  
s  
void main()  
d  
yylex();  
y. y.  
Output:  
> lex prg1.l  
> gcc lex.yy.c  
> ./a.out  
int abc;  
key word Identifier separator.

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 vowels and consonants in a string.**

**Code**

5. Count number of vowels and consonants.

```
y. x
#include <stdio.h>
int vowel=0, cons=0; // global declarations
y. }
y. y.
a|e|i|o|u|A|E|I|O|U & vowel++;  

{a-zA-Z} & const++;  

In & printf ("y. d consonants(%d", cons);  

printf ("y. d vowels(%d", vowel);  

y. y.

Output:
Kristen. If we run our
5 consonants
1 vowels
```

**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

4. Count number of words in a sentence.

y. x

#include <stdio.h>

int c=0;

y. }

y. y.

[a-zA-Z]\* x c++; }.

in printf ("v.d", c);

y. y.

Output:

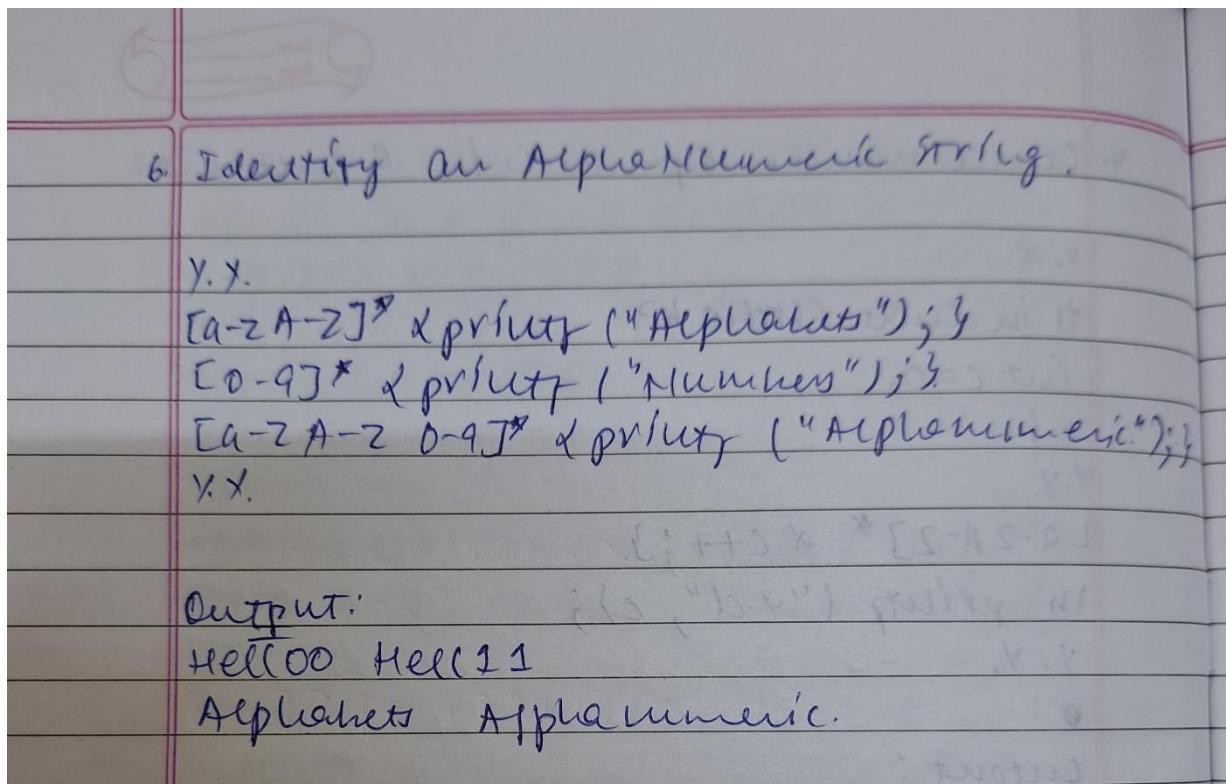
Hello bms college.

Output

```
Enter a sentence:  
This is compiler design lab work.  
No of words in the sentence are 6.  
The sun rises in the east and sets in the west.  
No of words in the sentence are 11.
```

## 2.2 Write a program in lex to demonstrate regular definition.

Code



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

7. Read input from file and print tokens on terminal.

X. Y.

Rules section from previous

Y. Y.

void main ()

d

char fname[10];

printf ("Enter name of file ");

scanf ("%s", fname);

yyin = fopen (fname, "r");

yylex();

fflush (yyin);

y.

Output:

Enter name of file file1.txt

Alphabets Alphabets Alphanumeric Numbers

## Output

The screenshot shows a terminal window with the following interface elements:

- Top bar: "Open" dropdown, a "+" button, and the file path "\*input.txt ~/Documents".
- Tab bar: "\*input.txt" (highlighted with a red underline) and "Week2\_fileInputFileOutp".
- Text area:

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

```
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**

8. Read input from file and print tokens into another file.

y. x.

Rules section from previous.

v. y.

void main()

x

char fname[10], oname[10];

printf("Enter name of the file ");

scanf ("%s", fname);

printf("Enter output file ");

scanf ("%s", oname);

yyin = fopen(fname, "r");

yyout = fopen(oname, "w");

yylex();

fclose(yyin);

fclose(yyout);

}

Output:

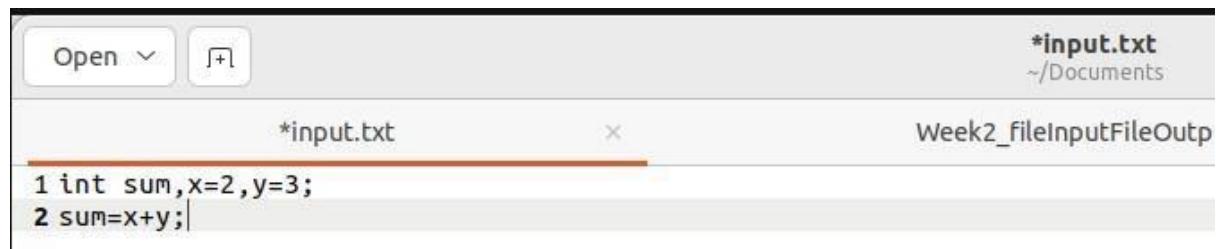
Enter name of file file1.txt

Enter output file output.txt

~~// the details accordingly to rules printed into  
output file.~~

S  
20/12/23

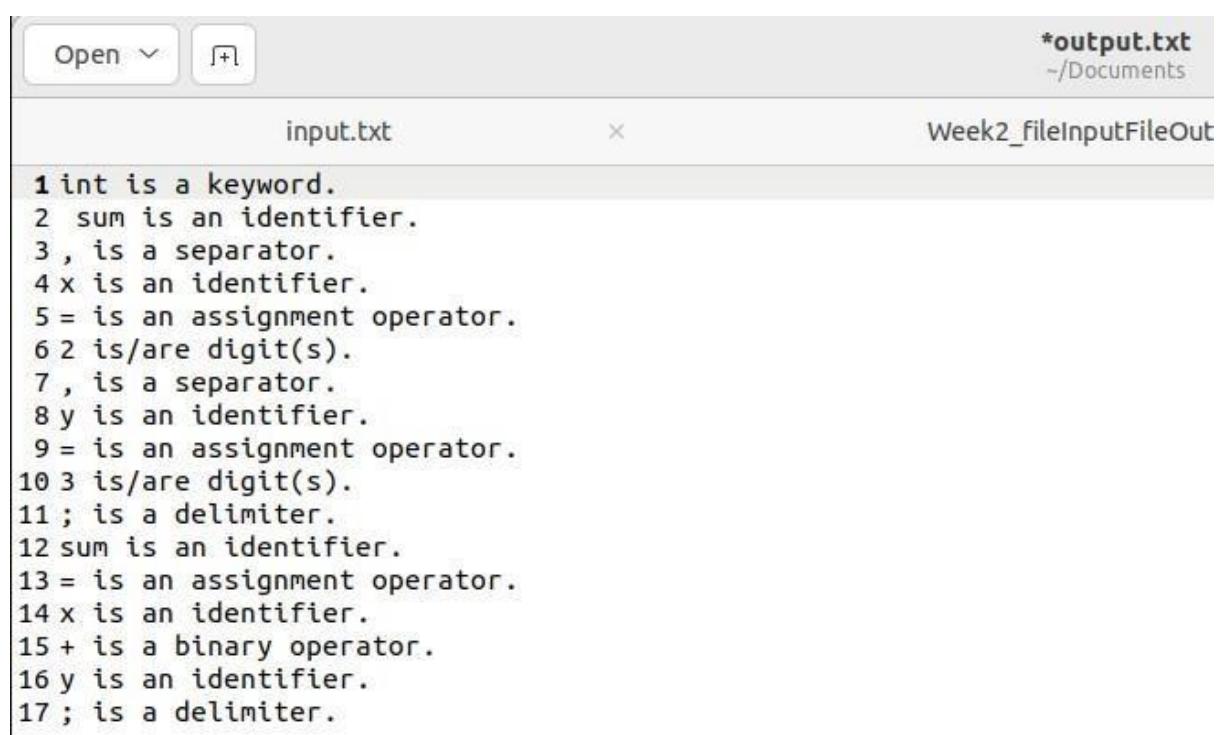
## Output



A screenshot of a text editor window. The title bar shows the file name as \*input.txt and the path as ~/Documents. The main area contains the following code:

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

Printed in output.txt



A screenshot of a text editor window. The title bar shows the file name as input.txt and the path as ~/Documents. The main area contains the following text, which appears to be the output of a lexical analyzer or parser:

```
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.
```

## Lab 3

3.1 Write a program in LEX to recognize Floating Point Numbers.

Code

```
6. LAB PROB. 1  
Identify floating point numbers.  
y. option noyywrap  
y. x  
#include <stdio.h>  
y. 3  
y. y.  
[+-]? [0-9]* [.] [0-9]+ &printf ("floating  
point numm");  
[+-]? [0-9]* &printf ("Not floating point");  
y. x.  
void main()  
&  
yylex();  
}
```

Output:

-23.67

floating point numm.

23.67

Floating point numm.

23

Not floating point numm.

23.

NOT floating point numm.

## 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

```
1. Identify simple and compound sentence  
y. option wrap  
y. #include <stdio.h>  
int a = 0;  
y. 3  
y. x.  
and | or | nevertheless | but | because | if | then  
{  
    a = 1;  
    [a - 2A - 2] * x 3  
    \n & return 0;  
y. x.  
void main()  
{  
    printf ("Enter the sentence ");  
    yylex();  
    if (a == 1)  
        printf ("Compound sentence ");  
    else  
        printf ("Simple sentence ");  
}
```

Output :  
And.  
Compound sentence.  
And  
Simple sentence.

## Output

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

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

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

**Code**

3. Ends with ., !, ?

V. Y.

```
[.?!] $ & a=1; b  
[ a-zA-Z ]* & 3  
in & return 0; }.
```

Output

Fields.

Ends with punctuation

They

Doesn't end with punctuation.

**Output**

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

```
Enter a sentence:  
Amazing!  
Ends with a punctuation!
```

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

**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**

2. Starts with articles.

```
y.y.  
^ (a|A|an|An|AN|the|The(THE) &a=1;b  
[a-zA-Z]*d  
In & return 0;b  
y.x,  
void main()  
{  
    printf ("Enter sentence");  
    yy+ex();  
    if (a==1)  
    {  
        printf ("Starts with article\n");  
    }  
    else  
    {  
        printf (" Doesn't start with article\n");  
    }  
}  
  
Output:  
Hello.  
Doesn't start with articles  
An Hello.  
Starts with articles.
```

## Output

```
Enter a sentence:  
This is a good idea.  
Does not start with an article!  
  
Enter a sentence:  
Amazing experience!  
Does not start with an article!  
  
Enter a sentence:  
The sun rises in the east.  
Starts with an article!  
  
Enter a sentence:  
A book is lying on the table.  
Starts with an article!  
  
Enter a sentence:  
An apple a day keeps the doctor away.  
Starts with an article!
```

**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

Handwritten Lex code:

```
task 9  
no: Count number of comment lines.  
y.y  
t+ include <stdio.h>  
int c=0;  
y.y  
" \n* "[\n\t]*\t+ /*[\n\t]*\t+ *\t+) \n\t+ c++ ;  
" //". * \c++ ;  
. EOF;  
y.y.  
void main()  
{  
yyin = fopen ("input.txt", "r");  
yyout = fopen ("output.txt", "w");  
yylex();  
printf ("The number of comments are: %d\n",  
fclose (yyin);  
fclose (yyout);  
}
```

### 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
```

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

**Code**

4. Signed & Unsigned numbers.  
y. X.  
^ [ + - ] [ 0 - 9 ] + { a = 1 ; }  
[ 0 - 9 ] + { }  
\n & return 0 ; }  
y. X.

Output

12  
unsigned.  
-12  
signed  
+12  
Signed.

*EF 21/11/23*

**Output**

```
Enter a number:  
123  
Unsigned number!  
  
-123  
Signed number!  
  
+123  
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

LATB - 4

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

y. A

```
#include <stdio.h>
#include <string.h>
#include <stropts.h>
char str1[200];
y. y
y. x.
[ \n ] & fprintf(yyout, "%s\n", str1); str1[0] = '\0';
[ \t ] & fprintf(yyout, "%s", str1);
str1[0] = '\0'; fprintf(yyout, "%s", " ");
- strcat(str1, yytext);
<< EOF >> & fprintf(yyout, "%s", str1);
return 0;
```

y. X.

```
int main()
{
    char filename[100];
    printf("Enter name of file : \t");
    scanf("%s", filename);
    yyin = fopen(filename, "r");
    // printf("Enter name of file to copy : \t");
    // scanf("%s", filename);
    yyin = fopen(filename, "r");
    yylex();
}

int yywrap()
```

## Output

```
*text.txt
1 Hello      World
2 Welcome to      programming|
```

Printed!

```
Open  ↗
print.txt
~/Documents
1 Hello World
2 Welcome to programming
```

**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**

2. a) strings ending in 00.

y. y.

[0-9]\* [00] ; #printf ("string ending with  
00\n");

void return 0; ;

y. y.

void main ()

{

printf ("Extra Number");

yytext();

y

Output:

Extra Number.

+800

~~string ending with 00.~~

## 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

6  
39

2.b) strings with three consecutive 2s.  
text equal to which contains 3 consecutive 2s  
Y, V.  
[0-9]\*[2][2][2][0-9]\* print ("String  
contains 222 in ", j, 3  
in return 0, 3  
Y, X

Output:  
Enter Number  
3622295  
String contains 222.

##### Output

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

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

4.2.3 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

a.d) set of all strings beginning with 1,  
binary representation of an integer  
is congruent to zero modulo 5.

y. y.  
 $\{0,1\}^*$  for ( $i=y$  to  $y-1$ ;  $i>=0$ ; $i--$ )  
value = value + (y \* pow(2, i));  
j++;  
if (value % 5 == 0).  
flag = 1;  
y  
(n) return 0;

Output:  
~~1000 & 101~~  
success.

## Output

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

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

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

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

#### 4.2.4 The set of all strings such that the 10th symbol from the right end is 1.

##### Code

2e) 10th digit from right is 1.

digit [0-9]  
xy.

{digit}+ I & digit}----& digit} & priority ("').  
10th symbol from  
right end is 1 "yy + (x);  
& digit}+ & priority (" condition met  
satisfied"); }

Output:  
101068691745  
10th symbol from right end is 1.

##### Output

```
Enter a string:  
23123456123  
10th symbol from right is not 1.  
Enter a string:  
11234345236  
10th symbol from right is 1.
```

#### 4.2.5 The set of all four digits numbers whose sum is 9.

##### Code

Q. f) set of all numbers (4 digit) such that  
sum is 9  
digits [0-9].  
Y. V.  
4 digits & digit & digit & digit  
& for ( i = yyLength - 1 ; i >= 0 ; i-- )  
    value += (yy.charAt(i) - 48);  
    if (value == 9)  
        flag = 1;  
    if (flag == 1)  
        return 0;  
Y. X.

Output:  
1341  
success.

##### Output

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

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

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

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

Code

Q) Set of all four digital numbers, whom individual digits are in ascending order from left to right.

digits [0-9].  
Y. Y.

digit 4 digit 3 digit 2 digit 1

for (i=0; i < yy.length-1; i++)  
if (yy.charAt(i) > yy.charAt(i+1))  
flag = 0;  
if (flag == 0)  
[in] return 0;

Output:  
8 1 2 3 4  
Success!

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

18/12/23  
LAB - 5

Purs: write a program to design lexical analyzer in C/C++ (to recognise keywords, identifiers, numbers, operators and punctuations)

```
#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 < size of (Keywords) / size of (Keywords[i]); i++)
                if (strcmp (token, Keywords[i]) == 0)
                {
                    printf ("Keyword: %s\n", token);
                    isKeyword = 1;
                    break;
                }
        }
    }
}
```

3.  
 if (!isKeyWord)  
 {  
 printf ("Identifier : %s\n", token);  
 }  
 else if (strchr ("+-\*/=<>(),", token[0])  
 != NULL)  
 {  
 printf ("Punctuation / Operator : %s\n",  
 token);  
 }  
 else  
 token = strtok (NULL, " \t\n");  
 }  
 int main ()  
 {  
 char inputCode [] = "if ( n > 0 )  
 {  
 x = n ; y = -x ;  
 }  
 lexicalAnalysis (inputCode);  
 return 0;  
 }

Output:-

Keyword: if  
 Punctuation / Operator: /  
 Identifier: n

Number: 0

:

;

Identifier: x

Punctuation / Operator: ;

SS  
Reported

## 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

8/18/24  
LAB - 6

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

$S \rightarrow cAd$   
 $A \rightarrow ab \mid a$

```
#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')
        ind++;
    printf("Hello\n");
    match('a');
    match('b');
```

}

else.

}

printf ("Parsing failed\n")

exit (1);

}

int main()

{

printf ("Enter input string\n");

scanf ("%s", input);

s();

if (input [ind] == '\$')

printf ("Parsing successful");

else.

printf ("Parsing failed\n");

return 0;

}

Output.

Enter input string.

Carroll \$

Hello

Parsing failed.

Enter input string

Carroll \$

Hello

Parsing successful.

## Output

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

```
Enter a string:  
caad$  
Invalid 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 +, -, \*, /.

Code

Ques: Design a suitable grammar for a simple calculator.

→ highest priority & right  
→ second highest priority & left a  
\*, / have third highest and left  
+, - having least and left.

prog1.y

y. option no y no exp.

y. x

# include "y.tab.h"

y. y

y. x

[0-9]+ y y (val = atoi yy + at); return NUM; ;  
[t t];  
in return 0;  
. return yy + ext [0];  
y. x

prog1.y

y. x

# include <stdio.h>

y. y

y. token. NUM.

y. left '+' '-'

y. left '\*' '/'

y. left 'Y.'

y. right '^'

y. x

```

expr : print("Valid expression\n");
       print("Result :%.d\n", $1); return 0;
| e : e '+' e   | $1 = $1 + $3
| e : e '-' e  | $1 = $1 - $3
| e : e '*' e  | $1 = $1 * $3
| e : e '/' e  | $1 = $1 / $3
| e : e '^' e  | $1 = $1 % $3
| e : e '^n' e | $1 = $3 ^ $3
| NUM          | $1 = $1; $3
|
| yyerror()    |
| print("Enter arithmetic expr\n");
| yyparse();
| return 0;
|
| yyerror()    |
| print("Invalid expression\n");
| return 0;
|
| Result:14

```

## Output

```

Enter an arithmetic expression:
2++3-
Invalid expression!
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

LAD-7

Ques: String Match

stringmatch.l

y. <

```
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yyleval;
y. }
```

[aA] & yyleval = yytext[0]; return a; }

[bB] & yyleval = yytext[0]; return b; }

\\ & return NL; }

, & return yytext[0]; }

y. y.

int yywrap()

{

return 1;

}

stringmatch.y

YYERROR

y. <

```
#include <stdio.h>
#include <stdlib.h>
int yyerror(char *s);
yytext(void);
```

y. }

y. token A

y. token B

y. token NL

y. x.

```

5 and 9
and 9

sutr! A A A A A S B NL d printf ("Parsed using
the rule (a^n)b, n>=5. Invalid string!") n,
;
S: S A
|
;
V. Y.
void main()
{
    printf ("Enter a string!\n");
    yyparse();
}
if yyerror (clear *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!

### **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

ques! Syntax Tree

Date \_\_\_\_\_  
Page \_\_\_\_\_

```
syntaxtree.y
Y. X
# include <stdio.h>
# include <stdlib.h>
# include "y.tab.h"
extern int yyval;
Y. Y
[0-9]+ yylex = atoi(yytext); return digit;
[ \t];
[ \n] return 0;
. return yytext[0];
Y. X
int yyerror()
{
    /* handle error */
    return 1;
}

syntaxtree.y
Y. X
# include <math.h>
# include <cctype.h>
# include <stdio.h>
# include <stdlib.h>
# include <y.tab.h>
put yyerror (char *s);
int yytext (void);
struct treenode
{
    char val[10]; put (c, rc;
};

Y;
```

int find;

struct treeNode synTree[100];

void myPrintTree(int curInd);

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

y.3

V. Token digit

y. v.

S: E & myPrintTree(\$1);

;

E: E' + T & \$2 = mknode(\$1, \$3, "+")

| T & \$2 = \$1;

;

T: T \* F & \$2 = mknode(\$1, \$3, "\*");

| F & \$2 = \$1;

;

F: ( E ') & \$2 = \$2; 3

| digit & clear buf[10]; sprintf(myp; "Y.%d", yyval);  
\$2 = mknode(-1, 7, buf); 3

;

y. v.

int main()

{

Ind=0;

printf("Enter an expression\n");

yyparse();

return 0;

3

int yyerror(char \*s)

{

printf("NYT Error\n");

return 0;

3

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

{

strcpy (syn-tree [ind].val, val);

syn-tree [ind].lc = (c);

syn-tree [ind].rc = rc;

lnd + 1;

return lnd - 1;

}

void myprinttree (int cur - lnd)

{

if (cur - lnd == -1) return;

if (syn-tree [cur - lnd].lc == -1 & syn-tree  
[cur - lnd].rc == -1)

printf ("Digit Node → Index : %d, Value : %s\n",  
curlnd, syn-tree [curlnd].val);

else

printf ("Operator Node → Index : %d, Value : %s,  
Left child Index : %d, Right child Index : %d\n", curlnd, syn-tree [curlnd].val,

syn-tree [curlnd].lc, syn-tree [curlnd].rc);

myprinttree (syn-tree [curlnd].lc);

myprinttree (syn-tree [curlnd].rc);

}

### Output

Enter an expression:

$x^3 + 5^4$ .

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.

## 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

Ques: Infix to Postfix

InfixtoPostfix.y

y.2

```
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yyval;
```

y.3

```
y. y,
[0-9]+ yval = atoi(yytext); return num; }
E \t ; } in { return 0; }
. \t return yytext[0]; }

y.4.
int yywrap()
```

y.5

yywrap

InfixtoPostfix.y

y.6

```
#include <stdio.h>
#include <stdlib.h>
int yyerror (char error *s);
int yylex (void);
```

y.7

```
y. token num
y. left '+' '-'
y. left '*' '/'
y. left ')'
y. left '('
y. right '^'
yy.
```

```

s: e & putchar ("\\n");  

;  

e: e+'+'& putchar ("+");  

| e-'-'& putchar (" - ");  

| t  

| ;  

t: t'*'& putchar ("*");  

| t '/'& putchar ("/");  

| h.  

| ;  

f: 'f'& putchar ('f');  

| num & printf ("%d", f);  

| ;  

h: f '^' h & putchar (^);  

| f.  

| ;  

void main()  

{  

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

    getchar ();  

    if (yyerror (const char *s))  

    {  

        printf ("Invalid infix expression\\n");  

        return 0;  

    }  

    output:  

    Enter an infix expression:  

    2+3*8/4^3-3  

    2 3 8 ^ 4 3 ^ / + 3 -

```

## 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

Ques: Three address code.

addresscode.y

#include <stdio.h>  
#include <stdlib.h>  
#include "y.tab.h"  
extern int yyval;  
extern char iden[20];  
yy  
d [0-9]+  
a [a-zA-Z]+  
y.y.  
if y & yyval = atoi(yytext); return digit;  
if y & strcpy(iden, yytext); yyval = 1; return id;  
else d;  
In return 0;  
. return yytext[0];  
v.y.  
if not yywrap()  
x  
return 1;  
y  
addresscode.y  
y.y.  
#include <math.h>  
#include <ctype.h>  
#include <stdio.h>  
int yyerror (char \*s);  
int yylex (void);  
int varint = 0;  
char iden [20];  
y.y.

y. token id

y. token digit

y. y.

S: id '=' E & printf ("y.s = %y.d\n", iden, varcut+1);  
 E : E '+' T & { varcut++; printf ("%ty.d =  
 %y.d + %y.d\n", \$9, \$1, \$3); }

+ E '-' T & { varcut++; printf ("%ty.d =  
 %y.d - %y.d\n", \$8, \$1, \$3); }

| T < \$9 = \$1; }

T: T '\*' F & { varcut++; printf ("%ty.d =  
 %y.d \* %y.d\n", \$9, \$1, \$3); }

| T '/' F & { varcut++; printf ("%ty.d =  
 %y.d / %y.d\n", \$8, \$1, \$3); }

| F & { \$9 = \$1; }

F: P '^' F & { varcut++; printf ("%ty.d =  
 %y.d\n", \$9, \$1); }

y.y.

~~int main()~~

x

varcut = 0;

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

yyparse();

return 0;

}

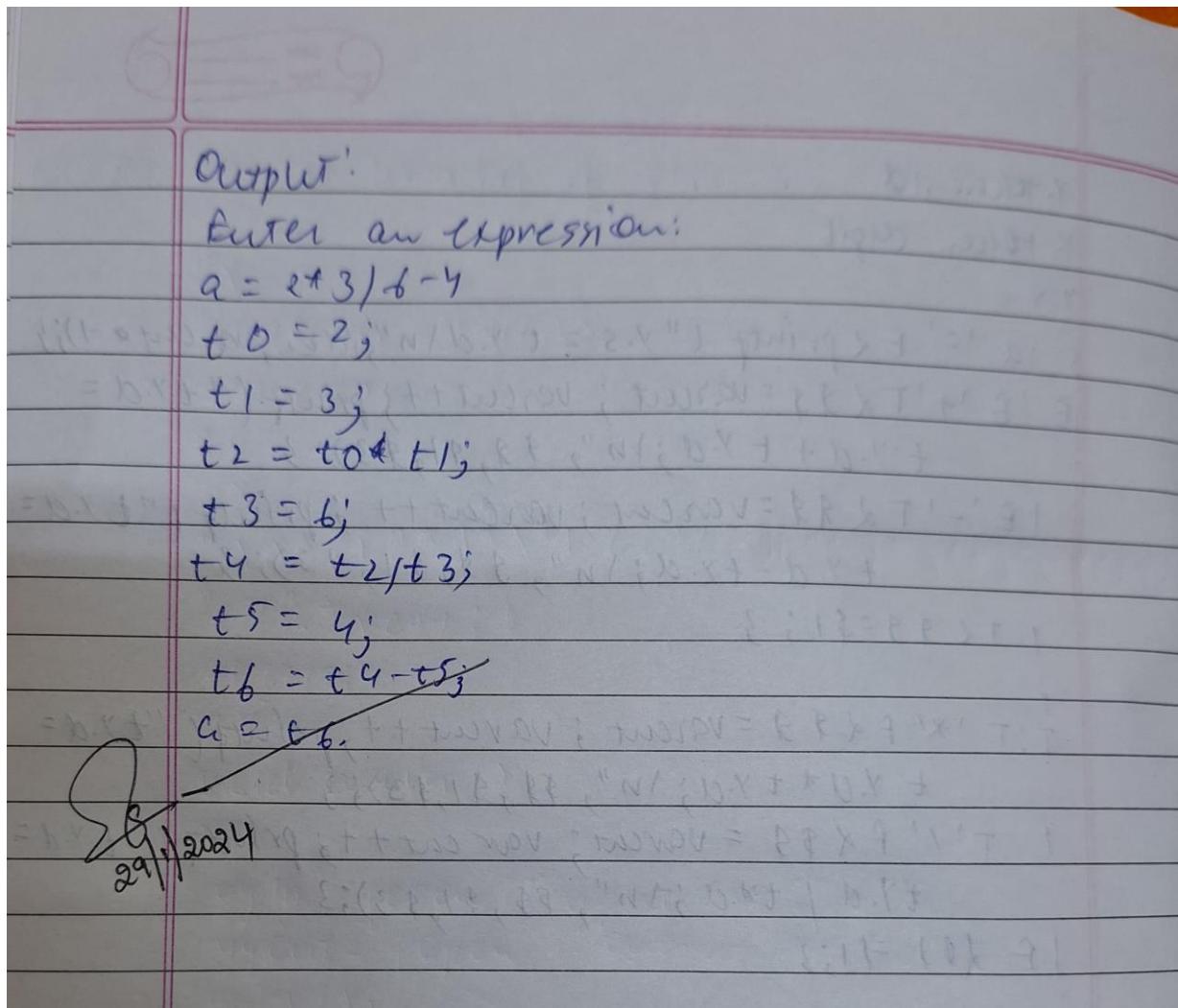
int yyerror (char \*s)

^

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
```