

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT

on

## COMPILER DESIGN

*Submitted by*

**Harsh Kumar (1BM21CS261)**

*Under the Guidance of*  
**Prof. Latha NR**  
**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 **Harsh Kumar (1BM21CS261)**, who is bona fide 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. Latha NR  
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, Harsh Kumar (1BM21CS261), 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. Latha NR, 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>Page No</b>
<b>1</b>		<b>6-9</b>
1.1	Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.	6-7
1.2	Write a program in LEX to count the number of characters and digits in astring.	8
	Write a program in LEX to count the number of vowels and consonants in astring.	9
<b>2</b>		<b>10-18</b>
2.1	Write a program in lex to count the number of words in a sentence.	10
2.2	Write a program in lex to demonstrate regular definition.	11-12
2.3	Write a program in lex to identify tokens in a program by taking input from afile and printing the output on the terminal.	13-14
2.4	Write a program in lex to identify tokens in a program by taking input from afile and printing the output in another file.	15-17
2.5	Write a program in lex to find the length of the input string.	18
<b>3</b>		<b>19-30</b>
3.1	Write a program in LEX to recognize Floating Point Numbers.	19-20
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 compoundelse it is simple.	21-22
3.3	Write a program to check if the input sentence ends with any of the followingpunctuation marks ( ?, fullstop , ! )	23-24
3.4	Write a program to read an input sentence and to check if the sentence beginswith English articles (A, a,AN,An,THE and The).	25-26
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 printthe count in a file called output.txt.	27-28
3.6	Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.	29-30
<b>4</b>		<b>31-44</b>
4.1	Write a LEX program that copies a file, replacing each nonempty sequence ofwhite spaces by a single blank.	31-32
4.2	Write a LEX program to recognize the following tokens over the alphabets {0,1,...,9}	33-44

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>		<b>45-48</b>
5.1	Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.	
<b>6</b>		<b>49-50</b>
6.1	Write a program to perform recursive descent parsing on the following grammar: S- >cAd A->ab   a	
<b>7</b>		<b>51-59</b>
7.1	Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.	51-53
7.2	Write a program in YACC to recognize strings of the form $\{(a^n)b, n \geq 5\}$ .	54-55
7.3	Write a program in YACC to generate a syntax tree for a given arithmetic expression.	56-59
<b>8</b>		<b>60-62</b>
8.1	Write a program in YACC to convert infix to postfix expression.	
<b>9</b>		<b>63-66</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:

Q1P  
Enter input  
good morning  
good is characters morning is character

1. if  
is us digits  
g O g I h  
go is characters : g I us digits  
h is characters.

\* if all character is from [a-zA-Z]  
will get individual line of output  
for each character

Q2) input : int a,b,(g) main()  
%{  
#include <stdio.h>  
%.y  
int float char & pnumf ("%.8 is keyword",  
"text");  
[a-zA-Z]\* ; /\* Identifier \*/  
%%.y  
%ywrap() // AS : Enter there is no  
more input given

int main()

{ printf("Enter input : "); }

scanf("%c", &c);

return 0;

}

O/P

Enter a, b, c

a is keyword

b is identifier

, is separator

c is identifier

, is separator

c is identifier

, is separator

Q3) count words & digits :-

O/P

Enter i/p Roll No & O/P

Count of words : 2 Count of numbers

: 1

Q4) vowels and consonants :-

Input : good

Count of vowels : 2

## 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**

Q3) count words & digits :-  
OIP  
anilu ijp Roll No 204  
count of words :- 2 count of numbers  
                  ! 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**

Q4) vowels and consonants :-  
Input? good  
Count of vowels ? 2  
Count of consonants ? 2

Code :-  
% { main ( ) {  
    char ch : input below is printing  
    alelilolol/A/E/I/O/U < C++ ; }  
    { a-z A-Z } { d++ ; }  
    if ( p == y ) {  
        count\_vowels++;  
        cout << "Count of vowels : " << count\_vowels;  
    }  
    else {  
        count\_consonants++;  
        cout << "Count of consonants : " << count\_consonants;  
    }  
} %

**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
```

## Lab 2

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

Code

```
%%
[ ~ \t \n ] + { words ++
\n { printf (" no of words is
% d ", words ),
words = 0; }
%%
```

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

```
Program 6.1  
0.6) recent valid string for alphabetical  
string and invalid not for alpha-  
numerical string.  
⇒ %  
    include < stdio.h >  
    %y  
    %%  
    [a-zA-Z]* { printf ("%s is valid  
    string ", yytext); }  
    [a-zA-Z0-9]* { printf ("%s is invalid  
    string ", yytext); }  
    %y  
    with yywrap()  
    {  
        and main()  
        {  
            printf ("Enter the sentence ");  
            yylex();  
            return 0;  
        }  
    }
```

O/P  
Harsh 123  
invalid string  
Harsh  
valid string

## 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**

```
Q7) Program to take input from a file and  
print output.  
/*  
 * include <stdio.h>  
 */  
int/char/float {printf ("In %s → keyword",  
yytext);}  
/*  
 * printf ("In %s → separator", yytext);  
 */  
⇒ {printf ("In %s → assignment operator, " yytext);}  
[0-9]* {printf ("In %s → digit", yytext);}  
[a-zA-Z0-9]* {printf ("In %s → identifier",  
yytext);}  
int wrap()  
{  
    /*  
     * yyin = fopen("input.txt", "r");  
     */  
    if (main ())  
    {  
        printf ("Enter ");  
        yyin = fopen ("input.txt",  
                    "r");  
        yylex();  
        return 0;  
    }  
}
```

## Output

```
*input.txt
~/Documents
*input.txt
Week2_fileInputFileOutput

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**

```
/* take input from input.txt and  
output to output.txt */  
%option noyywrap  
%include <stdio.h>  
%y %y  
yychar float { fprintf(yyout, "In %s->keyword",  
yytext); }  
; { fprintf(yyout, "In %s->separator", ); }  
= { fprintf(yyout, "In %s->assignment",  
yytext); }  
[0-9]* { fprintf(yyout, "In %s->digit",  
yytext); }  
[a-zA-Z0-9]* { fprintf(yyout, "In %s->identifier",  
yytext); }  
% %  
yywrap()  
{  
}
```

```

    main()
    {
        FILE *fin, *fout;
        fin = fopen("input.txt", "r");
        fout = fopen("output.txt", "w");
        char c;
        while ((c = fgetc(fin)) != EOF)
        {
            if (c == ' ')
                continue;
            else if (c >='0' & c <='9')
                fputc(c, fout);
            else if (c == '+')
                fputc('+', fout);
            else if (c == '-')
                fputc('-', fout);
            else if (c == '*')
                fputc('*', fout);
            else if (c == '/')
                fputc('/', fout);
            else if (c == '=')
                fputc('=', fout);
            else if (c == ',')
                fputc(',', fout);
            else if (c == ' ')
                continue;
            else
                fputc(' ', fout);
        }
    }

```

## Output

The screenshot shows a file editor window with two tabs: "input.txt" and "Week2\_fileInputFileOutput". The "input.txt" tab is active, displaying the following code:

```

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

```

Printed in output.txt

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

- Top bar: "Open" dropdown, a new file icon, and the file path "\*output.txt ~/Documents".
- Tab bar: "input.txt" and "Week2\_fileInputFileOut".
- Text area:

```
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**

```
%{  
[a-zA-Z 0-9 ! ? \t]+  
{ printf("length of  
input string is %d",  
yylen);}  
%}
```

**Output**

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

## Lab 3

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

#### Code

Week-3 project

1) Write a program in LEX to recognize floating point numbers. Check for all following input cases.

```
% {  
#include <stdio.h>  
%.  
%%.  
[0-9] + \. [0-9] + pexpr ("floating point  
number\n"),  
[0-9] + pexpr ("NOT a floating point  
number\n"),  
%{  
int yywrap()  
{  
}  
}  
int main()  
{  
    pexpr ("enter the sentence ");  
    yylex();  
    return 0;  
}
```

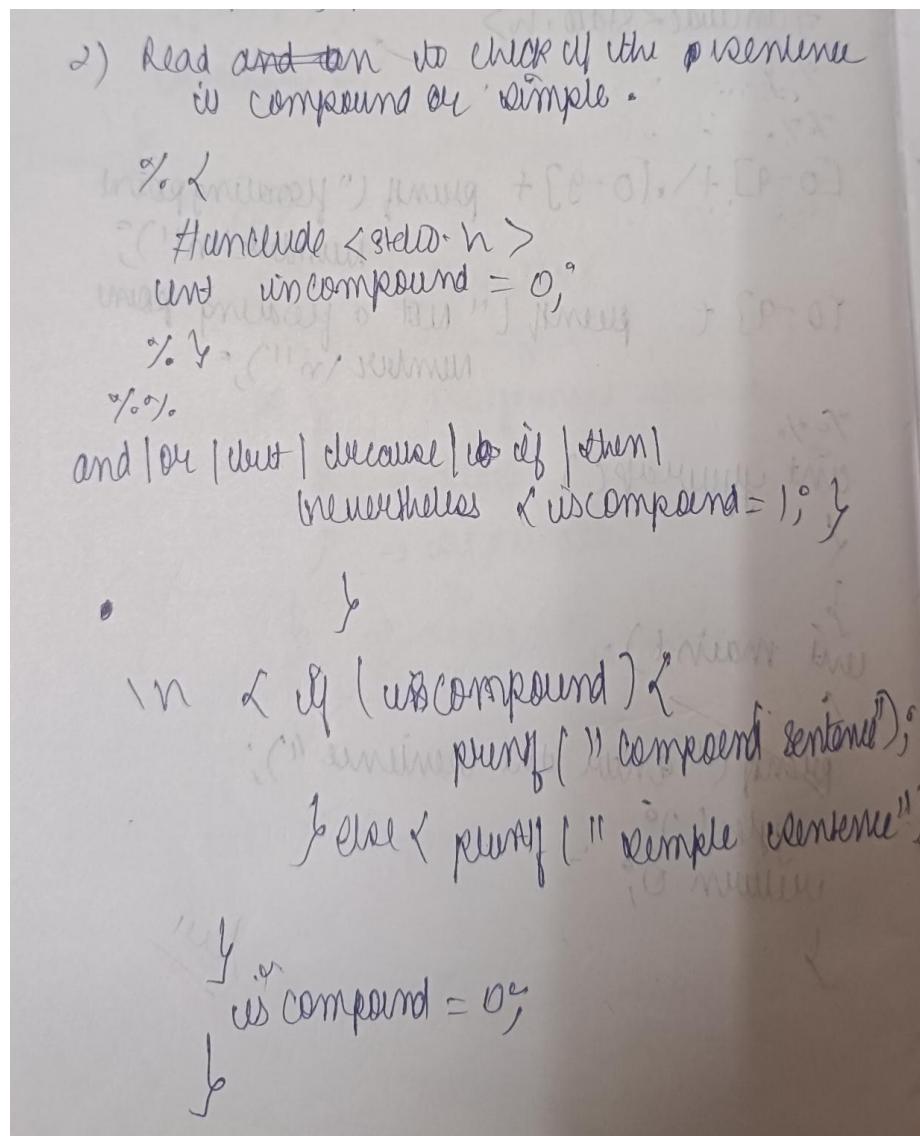
O/P  
Enter the number  
23.6  
floating point number  
45  
Not a floating point Number

## 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**



Y. %  
ant curmap()  
{  
}  
W main()  
{  
 print("Enter a sentence")  
 yylex();  
 return 0;  
}  
O/P  
Haus and Icumar  
Compound 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) write a C program to check if  
sentence ends with any of the  
punctuation mark.

→ %.  
#include <stdio.h>  
%

int ymwrap();  
{  
 char ch;  
 ch = ymwrap();  
 if (ch == '?' || ch == '!' || ch == '.'){  
 return 1;  
 } else {  
 return 0;  
 }  
}

%.  
\*.\*[?!.]\*{printf ("The sentence ends  
with punctuation mark");}  
{printf ("The sentence does not end  
with punctuation");}  
%

int ymwrap();  
{  
 char ch;  
 ch = ymwrap();  
 if (ch == '?' || ch == '!' || ch == '.'){  
 return 1;  
 } else {  
 return 0;  
 }  
}  
return 0;  
}

O/P  
Enter the sentence  
Hello!.

## 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**

```
Write a C program to check if a string starts with an article or not

⇒ %f
#include <stdio.h>
int flag = 0;
%y
%y.
AT|A|a|AN|an|THE|the| " " {flag=1;}
%y.
int yywrap()
{
    main()
    {
        printf("Enter a string ");
        yylex();
        if(flag == 1)
            printf("String starts with an article ");
        else
            printf("String does not start with an article ");
    }
    O/P
    An apple Yes
    After NO
```

## 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:  
An apple a day keeps the doctor away.  
Starts with an article!  
  
Enter a sentence:  
A book is lying on the table.  
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

6) Lex program to count no of  
comment lines in this no show strin

```
%{  
#include <stdio.h>  
int c = 0;  
char fname[20], bname[20];  
%}  
%  
%% /* [a-zA-Z0-9]*\n*/  
/* */  
/* [a-zA-Z0-9]+\n*/  
/* */  
int yyread()  
{  
    word main()  
    {  
        return FILE *yyin, *yyout;  
        scanf("y.s", fname);  
        scanf("y.s", bname, "%s");  
    }  
}
```

```
yyin = fopen(fname, "r");  
yyout = fopen(bname, "w");  
yylex();  
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  
^C
```

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

**Code**

4) write a C program to check whether the entered number is signed or unsigned

```
% {  
    #include <stdio.h>  
%.  
% %.  
[-+]? [0-9]+lp printf ("signed number\n");  
:[0-9]+.lp printf ("unsigned number\n");  
[a-zA-Z0-9]*lp printf ("invalid  
input\n");  
% %.  
int main()  
{  
    // O/P  
    // +2s → signed no  
    // 2s → unsigned no  
    // hello → invalid .  
}
```

(Followed by handwritten notes:  
1. Main function  
2. Input  
3. Output  
4. If +2s → signed no  
5. If 2s → unsigned no  
6. If hello → invalid .)

## Output

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

## Lab 4

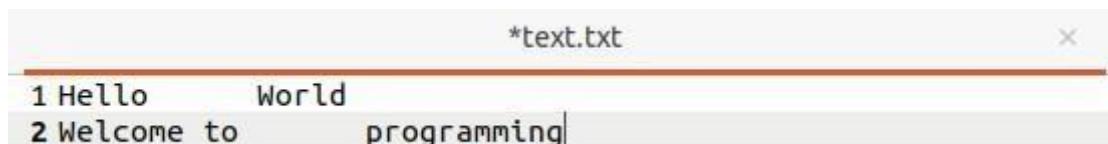
4.1 Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.

Code

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

```
→ %l
#include <stdio.h>
%j
y-y.
{if (" ") + fputc(yout, " ");
· | \n fputc(yout, "y.s"), ytext);
%y
int yywrap()
{
    return 1;
}
int main(void)
{
    yyin = fopen("input.txt", "r");
    yyout = fopen("output.txt", "w");
    yylax();
    return 0;
}
```

## 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) Write a LEX program to recognize  
the set of strings ending with 00

~~[0-9]\* 00 {~~  
~~yylex ("valid string: %s\n", yytext);~~

~~[0-9]\* 00 {~~  
~~yylex ("invalid string: %s\n", yytext);~~

ZIP  
1901 → invalid string  
1900 → valid string

## 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

```
* set of all string with 3 consecutive  
  222  
% %.  
* 222.*  
  many (" valid string & %&n",  
    ytext)  
} (y) have most else  
oo {D09) K. missed & 01  
printf (" invalid setting : %s  
    n", ytext);  
}  
} oo *{C-O}  
if (%c; 01P ;  
 2 2 2  
  value missing : 2 2 2  
  ) oo *{C-O}  
"m {2.3; missed setting  
  invalid setting : 12.3
```

##### 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 string such that every block of five consecutive symbols contains at least two 5's.

Code

```
c) /*  
 * If 5's flag = 0;  
 * for (i=0; i<5; i++)  
 {  
     int c = yytext[i] - '0';  
     if (c == 5)  
         count++;  
     if (count == 2)  
     {  
         flag = 1;  
         break;  
     }  
 }  
 if (flag != 1)  
 {  
     printf("invalid string");  
 }  
 */
```

## Output

```
Enter a string:  
1525558566  
yytext:15255,flag(1 if no of 5 is atleast 2):1  
yytext:8566,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!  
  
Enter a string:  
5432512345  
yytext:54325,flag(1 if no of 5 is atleast 2):1  
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

\* Set of all strings beginning with 1  
binary representation of an integer  
is congruent to zero modulo 5

#include <iostream>  
#include <string>

```
int main() {
    std::string s;
    std::cin >> s;
    int value = 0;
    int flag = 0;
    for (int i = s.length() - 1; i >= 0; i--) {
        value = value + (s[i] - '0') * pow(2, i);
        if (value % 5 == 0) {
            flag = 1;
        }
    }
    if (flag == 1) {
        std::cout << "Success" << std::endl;
    } else {
        std::cout << "Failure" << std::endl;
    }
}
```

OP

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.
```

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

##### Code

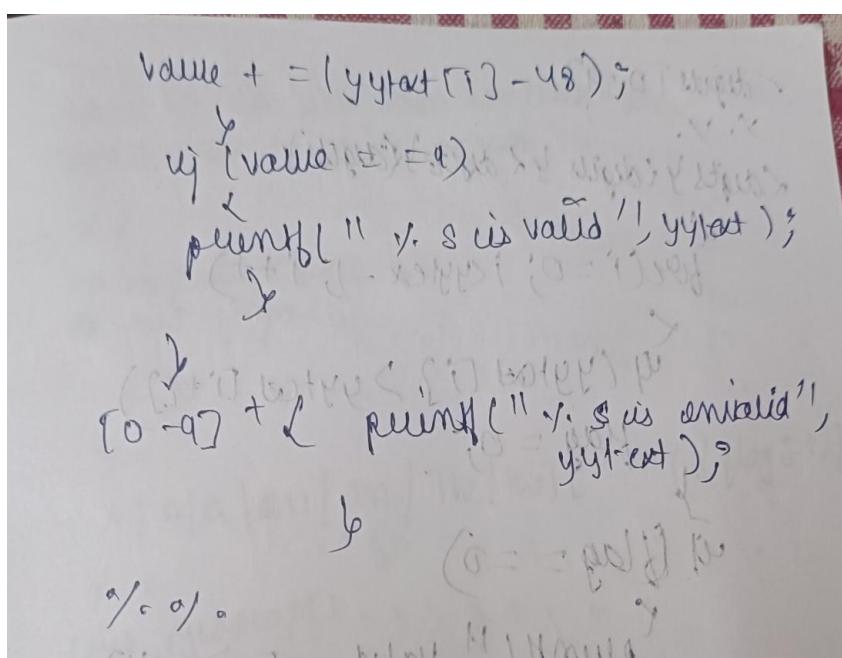
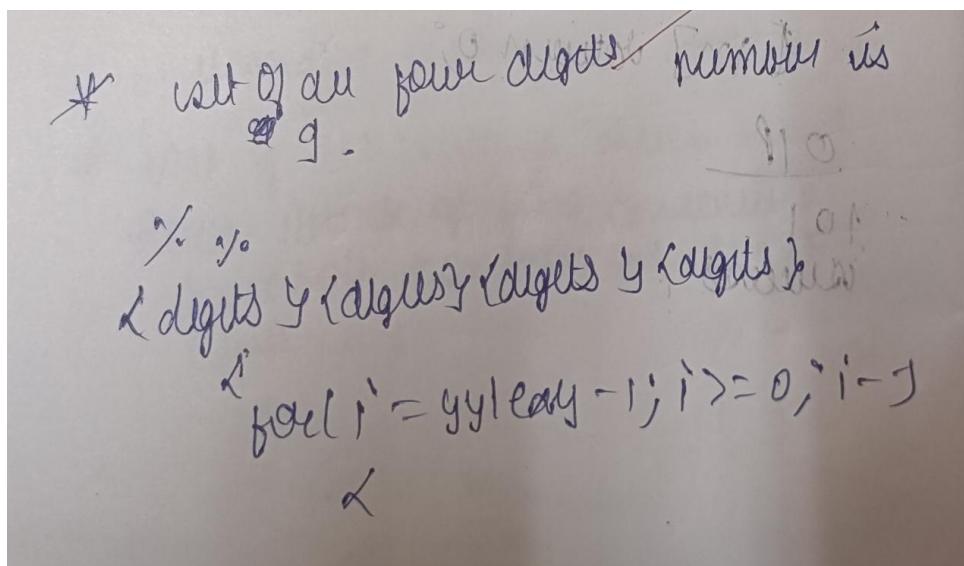
\* set of all strings such that 10<sup>th</sup> symbol from right end is 1  
→ digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }  
    %  
    {  
        { digits }  
        { digits }  
    } ( "y. S. 10<sup>th</sup> symbol  
        from right end is 1",  
        yheat );  
    { digits }  
    { & printf " condition  
        not satisfied" ); }  
    %  
    %

##### Output

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



## 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.
```

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

Code

```
    digits [0-9] M -> L37 twytyp. = 4 digit  
x, y.  
Length y < digits y < digits y < digits y  
for (i = 0; i < y + ex - 1; i++) {  
    if (yytow[i] > yytow[i + 1])  
        flag = 0;  
    else if (flag == 0)  
        printf(" Valid string ");  
    }  
else printf(" Invalid string ");  
} 0-9 * between 0;  
%o, %.
```

## 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

```
Program to design Lexical Analyzer in  
C Language
```

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

int iskeyword (char * str) {
    char keywords[5][10] = {"if", "else",
                           "while", "end", "return"};
    int i;
    for (int i = 0; i < 5; i++) {
        if (strcmp(str, keywords[i]) == 0)
            return 1;
    }
    return 0;
}

int isoperator (char ch) {
    char operators[] = "+-*";
    int i;
    for (i = 0; i < strlen(operators); i++)
        if (ch == operators[i])
            return 1;
    return 0;
}
```

```

    und isNumber (char *str)
    {
        int i = 0; // index of str
        for (i = 0; i < strlen(str); i++)
        {
            if (!isdigit(str[i])) {
                cout << "not a number" << endl;
                return;
            }
        }
        cout << str << endl;
    }

void analyze (char *input)
{
    char buffer[20];
    int i, j = 0;
    for (int i = 0; i < strlen(input); i++)
    {
        if (isoperator(input[i]) || // is punctuation (input[i])
            isSpace(input[i]) || // input[i] == ' '
            input[i] == '\0') // buffer[j] = '\0';
        {
            if (j > 0) {
                if (iskeyword(buffer)) {
                    printf ("Keyword: %s\n",
                           buffer);
                }
            }
        }
    }
}

```

```
else if (isNumber(buffer)) {  
    pennf("%d", buffer);  
}  
else if (isIdentifier(buffer)) {  
    pennf("%s", buffer);  
}  
  
if (isOperator(input[i]) || isPunct  
    isPunctuation(input[i])) {  
    pennf("operator/punctuation:  
        %c\n", input[i]);  
}  
  
if (j > 0) {  
    else {  
        buffer[j] = input[i];  
    }  
}
```

~~top~~

Unit main () {  
 char & input [100];  
~~print~~ (input);  
 print ("Enter the input string : ");  
 gets (input, sizeof (input), stdin);  
 input [sizeof (input) - 1] = '\0';  
 analyze (input);  
 return 0;  
}

QTP (QTP = QBasic / PASCAL - using) :-  
Enter the code ? and a sub, ~~sub~~  
Lexeme : int      Type : ~~key~~ keyword  
Lexeme : a      Type : identifier  
Lexeme : /      Type : punctuation  
Lexeme : b      Type : identifier  
~~Lexeme~~

See 19/12/23

## 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

PFA the program whatever do the  
program to perform descent parsing  
on following grammar at right side

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

$\Rightarrow$  include <stdio.h>  
int index = 0;  
void parse (char str[3]) {

if (input str[index] == 'a') {  
    index++;  
    if (input - str[1] == 'b') {  
        index++;  
        else if (input str[1] == 'a') {  
            index++;  
            else {  
                printf ("input string is  
                not accepted");  
            }  
        }  
    }  
}

}

```

void pause_s( char input_str[] )
{
    if ( input_str[0] == 'c' )
        {
            cout << "Input string is accepted";
            pause_s( input_str );
        }
    else if ( input_str[0] == 'd' )
        {
            cout << "Input string is not accepted";
            cout << endl;
        }
    else
        cout << "Input string is not accepted";
}

int main()
{
    char input_string[] = "cabd";
    cout << endl;
    pause_s( input_string );
    return 0;
}

```

## Output

```

1.S->cAd
2.A->ab/a
Enter any string:cabd
String is accepted by the grammar
The productions used are
S -> cAd
A -> ab

...Program finished with exit code 1
Press ENTER to exit console.

```

## Lab 7

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

Code

Will a YACC program to generate syntax tree for given arithmetic expression

PL.y

%{  
#include "y.tab.h"  
extern yyval;  
extern yytext;

%%

[0-9]+ [yylval = atoi(yytext); return digit;]  
[ \t];  
[\n] return 0;  
.\* yytext[0];  
% % int yyerror();

PL.y

% to run digit  
 % deft 141 1  
 % deft 141 11  
 % digit 141  
 %  
 % see L my-prime-due(\$1);  
 %  
 e: e 141 T L \$ = mnode(\$1, \$3, "1");  
 1 E 141 T L \$ = mnode(\$1, \$3, "1");  
 1 T L \$ = \$1;  
 % we have \$1 now  
 T: T 141 F L \$ = mnode(\$1, \$3, "1");  
 ( T 141 F L \$ = mnode(\$1, \$3, "1");  
 1 F L \$ = \$1;  
 f: ( (1 E 1) ) L \$ = \$2;  
 1 f 141 f L \$ = mnode(\$1, \$3, "1");  
 ( (1 digit (char diff 10), copyref (diff,  
 141 diff, y, val); \$ = mnode(1,  
 -1, deft);  
 31. (done now print-new) is  
 % % . 88 1 =>

## Output

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

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

Code

```
Yacc program to recognize strings  $\{a^n b | n > 5\}$ 
-----  

%code yyval
%{  

#include "y.tab.h";  

extern int yytext[10];  

%} % yyval = yytext[0];  

[a] { return A; }  

[aa] yyval = yytext[1];  

[b] { return B; }  

[n] { return N; }  

[bb] { return yytext[0], yytext[1]; }  

%left yywrap()  

int yywrap();  

[cc] { return 1; }  

}
```

p1.y  
 % Y. L  
 #include < stdio.h>  
 #include < stdlib.h>  
 main()  
 {  
 A B NL  
 % %  
 stmt : A A A A A S B NL  
 { printf ("valid string\n");  
 exit(0); }  
 ;  
 S : S A  
 ;  
 ;  
 % %  
 nt yyerror (char \* msg)  
 { printf (" Invalid string \n");  
 exit(0); }  
 Y  
 int main ()  
 { printf ("Enter the string \n");  
 yyparse();  
 }

## Output

```

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

Enter a string!
abc
Invalid String!
  
```

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

#### Code

Write a YACC program to generate syntax tree for given arithmetic expression

PL.y

```
%{  
#include "y.tab.h"  
extern void yyerror();  
%}  
%Y  
%N  
[0-9]+ [yyleval = atoi(yytext); return digit;]  
[\t];  
[\n] return 0;  
return yytext[0];  
% % use yywrap()  
%  
PL.y
```

```

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

struct tree_node {
    char val[10];
    int le;
    tree_node *left, *right;
};

tree_node *my_punkt_dree(int sind);
void my_punkt_dree(int sind);
int mknodc(int le, int sind, char val[10]);
}

% }

```

```

% to run digit
x. deft '+' ' '
y. deft '*' ' '
z. weight ' '
% %
$ e L my_punkt_dree($1); y
;
e = e ' + ' T L $$ = mknodc($1, $3, "+");
T = T ' * ' T L $$ = mknodc($1, $3, "-");
I F L $$ = $1;
I T L $$ = $1;
j
T = T ' * ' T L $$ = mknodc($1, $3, "*");
T = T ' / ' T L $$ = mknodc($1, $3, "/");
I F L $$ = $1;
I F L $$ = $2;
f = '((' E ')') L $$ = $2;
I f '^' f L $$ = mknodc($1, $3, "^");
I digit('char', deft[10]); exponent(deft,
    digit('char', deft[10]), exponen(deft,
    -1, deft));
% y.      g b = -->

```

```

% (def)
unt main()
{
    und = 0;
    puenf ("enter an expression");
    ypparse();
    und = und + 1;
    return 0;
}

unt ypparse()
{
    puenf ("NITW E-WOR");
    if (unt mode (unt 1%, unt dc,
        char val T10))
        astreep (syn-tree [und].val, val);
    syn-tree [und].dc = dc;
    syn-tree [und].rc = rc;
    und++;
    return und;
}

void print-tree (und cur-and)
{
    if (cur-and != -1)
        cellus;
    if (syn-tree [cur-and].dc
        == -1 &&

```

```

if n->type == "num-cnd" val = -1;
plumb ("digit node → whole : %d,
value: %s\n", cur-wnd, sym-
true[cur-wnd].val);

else plumb ("operator Node → wide : %c,
value : %c\n", cur-wnd, sym-
true[cur-wnd].val);
else my_plumb_true(sym_true[cur-wnd].val);
my_plumb_true(sym_true[cur-wnd].val);

y
    if number : (A)
O/P    width = 1010 - 1010
S + S * (2 - 4) ^ 2
operator Node → wide x: 3, value: +,
left-child = Digit Node → wide 0
rightNode → 7
operator Node → wide : -1 value: 2
left-child = digit Node : 5 right child under
= 8
2

```

## 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

push 2 % modify the programs so as  
to include operator  
as '++' '-' '++' as per their  
associativity and precedence.

P4.c

% L  
#include "y.tab.h"  
return yyval;

% } % % %  
[0-9]+  
yyval = atoi(yytext);  
return 'digit';

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

% %  
cent yywrap();

Y

14.4

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

%{

% token digit

% left '+' '-'

% right '\*' '/'

% right '^'

```
void E() { printf("E\n"); }
```

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

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

```
E: E '+' E L printf("E+\n"); }
```

```
; E: E '-' E L printf("E-\n"); }
```

```
E: E '*' E L printf("E*\n"); }
```

```
; E: E '/' E L printf("E/\n"); }
```

```
E: E '^' E L printf("E^\n"); }
```

```
; E: E '!' E L printf("E!\n"); }
```

```
E: ! C' E '!'
```

```
; E: digit L printf("digit\n"); }
```

}; %

```
cent main()
{
    cout << "Enter infix expression: ";
    cin >> str;
    cout << "Input string: " << str;
    cout << endl;
    cout << "Parsing expression: ";
    cout << str;
    cout << endl;
    cout << "Infix expression: ";
    cout << str;
    cout << endl;
    cout << "Postfix expression: ";
    cout << str;
    cout << endl;
    cout << "Prefix expression: ";
    cout << str;
    cout << endl;
    cout << "Evaluation of expression: ";
    cout << str;
    cout << endl;
}
```

cout << "Enter infix expression: ";
 cin >> str;
 cout << "Input string: " << str;
 cout << endl;
 cout << "Parsing expression: ";
 cout << str;
 cout << endl;
 cout << "Infix expression: ";
 cout << str;
 cout << endl;
 cout << "Postfix expression: ";
 cout << str;
 cout << endl;
 cout << "Prefix expression: ";
 cout << str;
 cout << endl;
 cout << "Evaluation of expression: ";
 cout << str;
 cout << endl;

## 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

23/12/24  
Week -9

Use YACC to generate 3-address code for  
the a given expression

```
PI-2. d[0-9] +.  
      a[a-zA-Z] +  
  
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <y.tab.h>  
extern char iden[20];  
  
%y.  
  
{ d{ lyy1val = atoi(yytext); return digit; }  
  [^c]*; }  
  
%return 0;  
%return yychar[0];  
  
%y.  
  
int yymain();  
{  
}
```

p1.y

y-f

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

int var_cnt = 0;
char iden[20];
```

y-j

```
S : id = E { printf("y-s = t y.d\n", iden, var_cnt - 1); }
```

```
E : E + T { $ = var_cnt++; printf("t y.d = t/d + t(y.d; \n", $, $1, $3); }
```

```
| E - T { $ = var_cnt++; printf("t y.d = t y.d - t y.d; \n", $, $1, $3); }
```

y

1 + f \$ - \$1; y

T : T \* F { \$ = var\_cnt++; var\_cnt++;

printf("t y.d = t y.d \* t y.d; \n", \$, \$1, \$3); }

| T / F { \$ = var\_cnt++; var\_cnt++; printf

("t y.d / t y.d; \n", \$, \$1, \$3); }

```

IF ($$ == $1;)
{
    if ($$ == '+') var_cnt++; var_cnt++;
    print(" + y.d = (" + d * t) / d; \n", $$, ($1, $2));
    LPH $$ == $1; y.d = y.d; \n", $$, ($1, $2));
}

if ($$ == '$2;')
{
    if ($$ == '+') var_cnt++; var_cnt++;
    print(" + y.d = (" + d * t) / d; \n", $$, ($1, $2));
    y.d = y.d; \n", $$, ($1, $2));
}

int main()
{
    var_cnt = 0;
    print("Enter an expression: \n");
    cin >> s;
    return 0;
}

```

Output

Enter an expression.

a = 2 + 3 \* 6.

t0 = 2;

t1 = 3;

t2 = 6;

t3 = t1 \* t2;

t4 = t0 + t3;

a = t4.

## 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
```