

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on COMPILER DESIGN

Submitted by

MAHAVIR NAHATA(1BM21CS100)

Under the Guidance of
Prof. Prameetha Pai 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 Mahavir Nahata(**1BM21CS100**) , 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. Prameetha Pai
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, Mahavir Nahata (1BM21CS100), 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. Prameetha Pai, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

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

TABLE OF CONTENTS

Lab No	Title	Page No
1		6-8
1.1	Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.	6
1.2	Write a program in LEX to count the number of characters and digits in a string.	7
	Write a program in LEX to count the number of vowels and consonants in a string.	8
2		9-15
2.1	Write a program in lex to count the number of words in a sentence.	9
2.2	Write a program in lex to demonstrate regular definition.	10
2.3	Write a program in lex to identify tokens in a program by taking input from a file and printing the output on the terminal.	11-12
2.4	Write a program in lex to identify tokens in a program by taking input from a file and printing the output in another file.	13-14
2.5	Write a program in lex to find the length of the input string.	15
3		16-23
3.1	Write a program in LEX to recognize Floating Point Numbers.	16
3.2	Read and input sentence, and check if it is compound or simple. If a sentence has the word- and , or ,but ,because ,if ,then ,nevertheless then it is compound else it is simple.	17
3.3	Write a program to check if the input sentence ends with any of the following punctuation marks (?, fullstop , !)	18-19
3.4	Write a program to read an input sentence and to check if the sentence begins with English articles (A, a,AN,An,THE and The).	20-21

3.5	Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt.	22
3.6	Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character.	23
4		24-36
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.
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.
5	
5.1	Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations.
6	
6.1	Write a program to perform recursive descent parsing on the following grammar: S->cAd A->ab a
7	
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>=5}.
7.3	Write a program in YACC to generate a syntax tree for a given arithmetic expression.
8	
8.1	Write a program in YACC to convert infix to postfix expression.

9	
9.1	Write a program in YACC to generate three address code for a given expression.

04/11/23

(*)

yywrap

lex (programname.l) → in cont ↳ i/p stream is
over

gcc lex.o yy.c

~~run.out~~ ./a.out

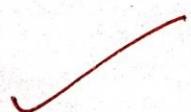
Q1) Write a lex program to identify datatype - int, char, float & variables;

```
%{  
#include <stdio.h>  
%}  
int | char | float { printf("%s is keyword", yytext); }  
[_-a-zA-Z] [_-0-9a-zA-Z]* { printf("%s is variable", yytext); }  
%  
int yywrap(){  
    return 0; }  
int main(){  
    yyFlex();  
    return 0;  
}
```

Output:

- int
int is keyword,
- char
char is keyword,
- float
float is keyword /
- int

- 1 int
1 is invalid token int is keyword.
- 1 is invalid token



② Write a lex program to identify each character as a consonant or vowel in a given sentence.

Y {

#include <stdio.h>

a b, l

Y }

Y x

~~Face your dad.~~

region A E I O U { printf ("%c is a vowel\n", y); exit(0) } .

[a-zA-Z] { printf ("%c is a consonant\n", y); exit(0) }

yywrap() {

return 0; }

int main() { -A s-a-e-o-] [e-q] [p-o]

{ yywrap(); exit(0) }

return 0; }

3

Output:

Compiler Design

No. of vowels and consonants are 5 and 9.

This is a ~~loop~~ ~~break~~ ~~break~~ at 3.

No. of vowels and consonants are 5 and 6

negative

{ () goes to 5 }

{ 0 goes to 6 }

for(;; i++)

{ () goes to 5 }

{ 0 goes to 6 }

if (i == 5)

else if (i == 6)

else if (i == 7)

else if (i == 8)

else if (i == 9)

else if (i == 10)

else if (i == 11)

else if (i == 12)

else if (i == 13)

else if (i == 14)

else if (i == 15)

else if (i == 16)

else if (i == 17)

else if (i == 18)

else if (i == 19)

else if (i == 20)

else if (i == 21)

else if (i == 22)

else if (i == 23)

else if (i == 24)

else if (i == 25)

else if (i == 26)

else if (i == 27)

else if (i == 28)

else if (i == 29)

else if (i == 30)

else if (i == 31)

else if (i == 32)

else if (i == 33)

else if (i == 34)

else if (i == 35)

else if (i == 36)

else if (i == 37)

else if (i == 38)

else if (i == 39)

else if (i == 40)

else if (i == 41)

else if (i == 42)

else if (i == 43)

else if (i == 44)

else if (i == 45)

else if (i == 46)

else if (i == 47)

else if (i == 48)

else if (i == 49)

else if (i == 50)

else if (i == 51)

else if (i == 52)

else if (i == 53)

else if (i == 54)

else if (i == 55)

else if (i == 56)

else if (i == 57)

else if (i == 58)

else if (i == 59)

else if (i == 60)

else if (i == 61)

else if (i == 62)

else if (i == 63)

else if (i == 64)

else if (i == 65)

else if (i == 66)

else if (i == 67)

else if (i == 68)

else if (i == 69)

else if (i == 70)

else if (i == 71)

else if (i == 72)

else if (i == 73)

else if (i == 74)

else if (i == 75)

else if (i == 76)

else if (i == 77)

else if (i == 78)

else if (i == 79)

else if (i == 80)

else if (i == 81)

else if (i == 82)

else if (i == 83)

else if (i == 84)

else if (i == 85)

else if (i == 86)

else if (i == 87)

else if (i == 88)

else if (i == 89)

else if (i == 90)

else if (i == 91)

else if (i == 92)

else if (i == 93)

else if (i == 94)

else if (i == 95)

else if (i == 96)

else if (i == 97)

else if (i == 98)

else if (i == 99)

else if (i == 100)

else if (i == 101)

else if (i == 102)

else if (i == 103)

else if (i == 104)

else if (i == 105)

else if (i == 106)

else if (i == 107)

else if (i == 108)

else if (i == 109)

else if (i == 110)

else if (i == 111)

else if (i == 112)

else if (i == 113)

else if (i == 114)

else if (i == 115)

else if (i == 116)

else if (i == 117)

else if (i == 118)

else if (i == 119)

else if (i == 120)

else if (i == 121)

else if (i == 122)

else if (i == 123)

else if (i == 124)

else if (i == 125)

else if (i == 126)

else if (i == 127)

else if (i == 128)

else if (i == 129)

else if (i == 130)

else if (i == 131)

else if (i == 132)

else if (i == 133)

else if (i == 134)

else if (i == 135)

else if (i == 136)

else if (i == 137)

else if (i == 138)

else if (i == 139)

else if (i == 140)

else if (i == 141)

else if (i == 142)

else if (i == 143)

else if (i == 144)

else if (i == 145)

else if (i == 146)

else if (i == 147)

else if (i == 148)

else if (i == 149)

else if (i == 150)

else if (i == 151)

else if (i == 152)

else if (i == 153)

else if (i == 154)

else if (i == 155)

else if (i == 156)

else if (i == 157)

else if (i == 158)

else if (i == 159)

else if (i == 160)

else if (i == 161)

else if (i == 162)

else if (i == 163)

else if (i == 164)

else if (i == 165)

else if (i == 166)

else if (i == 167)

else if (i == 168)

else if (i == 169)

else if (i == 170)

else if (i == 171)

else if (i == 172)

else if (i == 173)

else if (i == 174)

else if (i == 175)

else if (i == 176)

else if (i == 177)

else if (i == 178)

else if (i == 179)

else if (i == 180)

else if (i == 181)

else if (i == 182)

else if (i == 183)

else if (i == 184)

else if (i == 185)

else if (i == 186)

else if (i == 187)

else if (i == 188)

else if (i == 189)

else if (i == 190)

else if (i == 191)

else if (i == 192)

else if (i == 193)

else if (i == 194)

else if (i == 195)

else if (i == 196)

else if (i == 197)

else if (i == 198)

else if (i == 199)

else if (i == 200)

else if (i == 201)

else if (i == 202)

else if (i == 203)

else if (i == 204)

else if (i == 205)

else if (i == 206)

else if (i == 207)

else if (i == 208)

else if (i == 209)

else if (i == 210)

else if (i == 211)

else if (i == 212)

else if (i == 213)

else if (i == 214)

else if (i == 215)

else if (i == 216)

else if (i == 217)

else if (i == 218)

else if (i == 219)

else if (i == 220)

else if (i == 221)

else if (i == 222)

else if (i == 223)

else if (i == 224)

else if (i == 225)

else if (i == 226)

else if (i == 227)

else if (i == 228)

else if (i == 229)

else if (i == 230)

else if (i == 231)

else if (i == 232)

else if (i == 233)

else if (i == 234)

else if (i == 235)

else if (i == 236)

else if (i == 237)

3) Write lex program to identify characters, numbers or digits;

Expected output:

Enter input : sum 33

sum is stream of characters
33 is digit

{. int main() { char c; }

so 10
1. {

#include <stdio.h>

2. }

3. .

[a-zA-Z] * printf("y.%c is character", y); }
[0-9] * printf("y.%d is digit", y); }

4. .

int yywrap() {

return 0; }

int main()

{

yyflex();

return 0;

}

Output:

33

33 is digit

sum

sum is alphabet

sum 33

sum is character 33 is digit

3 completed

~~See
18/11/23.~~

4) Write C program to count no. of words in an i/p sentence.

y.c

#include <stdio.h>

y.c

y.c

[a-z A-Z 0-9] * { c = c + 1; }

if n { printf("%d\n", c); c = 0; }

y.c

int ywrep()

{ return 0; }

int main()

{

ywrrep();

return 0; }

output:

hello m

2

mahanir graham is a good boy

6

~~2 5 1 1 8~~

words in the sentence are 6.

Lab program:

5) write a program in LEX to recognize floating point numbers:

```
y. {  
#include <stdio.h>  
#include <string.h>
```

```
y. }  
y.%
```

```
[+-] [0-9] + [.] [0-9] + %printf("floating point no.");  
[+-]? [0-9] +
```

```
y. x
```

```
int wrap () {  
    return 0;  
}
```

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

Output:

```
+22.7  
floating point no.  
-23.7  
floating point no.
```

```
22.88  
floating point no.
```

✓ Dev 25/11/23

✓ Dev 25/11/23
2 programs solved

(no marks)

(0 marks)

: angle

1.00	0.00
1.1000	0.1000
1.0100	0.0100
1.0010	0.0010

✓ Dev 25/11/23

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

(a) set of all strings ending in "00",

$$\downarrow \quad (\text{circle}) [0-a]^* [0 \ 0]$$

2.3 "on the way to what?" Through the P-TEF-B complex

~~Includes estimate of the~~

#include <string.h>

× 3

二二

[0-9]^{*} [0] [0]

```
{ printf (" valid"); }
```

二、二

int y y wrap ()

{ return 0;

3 ~~Never~~

```
int main()
```

{ younger ()

return 0;

3

Output:

100

valid

001

Valid 1

001100

هـ ١٦٣

0011

Valid 11

~~Jan
26/12/23~~

1.1 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.
```

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

Cod::

```
% {  
#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.
```

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

Code::

```
%{  
#include<stdio.h> int flag=0;  
%}  
%%  
^(an|An|The|the|A|a)[ " ].* {flag=1;}  
..* {flag=0;}  
\n {return 0;}  
%%  
int yywrap()  
{  
    return 1;  
}  
void main()  
{  
    printf("Enter a sentence:\n"); yylex();  
    if(flag==1)  
        printf("Starts with an article!\n"); else  
        printf("Does not start with an article!\n");  
}
```

Output::

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

1.4 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 terminal window with two tabs open. The top tab is titled '*input.txt' and contains the following code:

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

The bottom tab is titled '*output.txt' and displays the tokens identified by the lexer:

```
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.
```

Q)Write a program in lex to find the length of the input string.

Code:

```
%{  
#include<stdio.h>  
%}  
%%  
[a-zA-Z0-9.,!?.\t]+ {printf("Length of input string is %d.\n",yyleng);}  
%%  
void main()  
{  
printf("Enter a string:\n");  
yylex();  
}  
int yywrap()  
{  
return 1;  
}  
Output:
```

```

neha29@neha-VirtualBox:~/Documents$ lex Week2_lengthofString.l
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a string:
Good Morning!
Length of input string is 13.

Where do you stay?
Length of input string is 18.

```

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

Code::

```

%{
#include<stdio.h>

int flag=0;
%}
%%

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

int yywrap()
{
return 1;
}

void main()

```

```

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

yylex();
if(flag==1)
printf("Compound sentence!\n");

else
printf("Simple sentence!\n");
}

```

Output

Enter a sentence:

This is a car.

Simple sentence!

Enter a sentence:

She is good at singing and dancing.

Compound sentence!

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

Code

```

%{

#include<stdio.h>

int c=0;

%}

% %

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

"/\\.* {c++;}

.ECHO;

% %

int yywrap()
{
return 1;

```

```
}

void main()
{
yyin=fopen("input.txt","r");

yyout=fopen("output.txt","w");

yylex();

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

fclose(yyin);

fclose(yyout);

}
```

Output:

```
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
```

(b) Set of all strings with three consecutive 2's

Logic: $[0-9]^* [2][2][2] [0-9]^*$

```
#include <iostream.h>
#include <string.h>
```

```
y.1
```

```
y.2
```

```
y.3
```

```
y.4
```

```
y.5
```

```
y.6
```

```
y.7
```

```
y.8
```

```
y.9
```

```
y.10
```

```
y.11
```

```
y.12
```

```
y.13
```

```
y.14
```

```
y.15
```

```
y.16
```

```
y.17
```

```
y.18
```

```
y.19
```

```
y.20
```

```
y.21
```

```
y.22
```

```
y.23
```

```
y.24
```

```
y.25
```

```
y.26
```

```
y.27
```

```
y.28
```

```
y.29
```

```
y.30
```

```
y.31
```

```
y.32
```

```
y.33
```

```
y.34
```

```
y.35
```

```
y.36
```

```
y.37
```

```
y.38
```

```
y.39
```

```
y.40
```

```
y.41
```

```
y.42
```

```
y.43
```

```
y.44
```

```
y.45
```

```
y.46
```

```
y.47
```

```
y.48
```

```
y.49
```

```
y.50
```

```
y.51
```

```
y.52
```

```
y.53
```

```
y.54
```

```
y.55
```

```
y.56
```

```
y.57
```

```
y.58
```

```
y.59
```

```
y.60
```

```
y.61
```

```
y.62
```

```
y.63
```

```
y.64
```

```
y.65
```

```
y.66
```

```
y.67
```

```
y.68
```

```
y.69
```

```
y.70
```

```
y.71
```

```
y.72
```

```
y.73
```

```
y.74
```

```
y.75
```

```
y.76
```

```
y.77
```

```
y.78
```

```
y.79
```

```
y.80
```

```
y.81
```

```
y.82
```

```
y.83
```

```
y.84
```

```
y.85
```

```
y.86
```

```
y.87
```

```
y.88
```

```
y.89
```

```
y.90
```

```
y.91
```

```
y.92
```

```
y.93
```

```
y.94
```

```
y.95
```

```
y.96
```

```
y.97
```

```
y.98
```

```
y.99
```

```
y.100
```

```
y.101
```

```
y.102
```

```
y.103
```

```
y.104
```

```
y.105
```

```
y.106
```

```
y.107
```

```
y.108
```

```
y.109
```

```
y.110
```

```
y.111
```

```
y.112
```

```
y.113
```

```
y.114
```

```
y.115
```

```
y.116
```

```
y.117
```

```
y.118
```

```
y.119
```

```
y.120
```

```
y.121
```

```
y.122
```

```
y.123
```

```
y.124
```

```
y.125
```

```
y.126
```

```
y.127
```

```
y.128
```

```
y.129
```

```
y.130
```

```
y.131
```

```
y.132
```

```
y.133
```

```
y.134
```

```
y.135
```

```
y.136
```

```
y.137
```

```
y.138
```

```
y.139
```

```
y.140
```

```
y.141
```

```
y.142
```

```
y.143
```

```
y.144
```

```
y.145
```

```
y.146
```

```
y.147
```

```
y.148
```

```
y.149
```

```
y.150
```

```
y.151
```

```
y.152
```

```
y.153
```

```
y.154
```

```
y.155
```

```
y.156
```

```
y.157
```

```
y.158
```

```
y.159
```

```
y.160
```

```
y.161
```

```
y.162
```

```
y.163
```

```
y.164
```

```
y.165
```

```
y.166
```

```
y.167
```

```
y.168
```

```
y.169
```

```
y.170
```

```
y.171
```

```
y.172
```

```
y.173
```

```
y.174
```

```
y.175
```

```
y.176
```

```
y.177
```

```
y.178
```

```
y.179
```

```
y.180
```

```
y.181
```

```
y.182
```

```
y.183
```

```
y.184
```

```
y.185
```

```
y.186
```

```
y.187
```

```
y.188
```

```
y.189
```

```
y.190
```

```
y.191
```

```
y.192
```

```
y.193
```

```
y.194
```

```
y.195
```

```
y.196
```

```
y.197
```

```
y.198
```

```
y.199
```

```
y.200
```

```
y.201
```

```
y.202
```

```
y.203
```

```
y.204
```

```
y.205
```

```
y.206
```

```
y.207
```

```
y.208
```

```
y.209
```

```
y.210
```

```
y.211
```

```
y.212
```

```
y.213
```

```
y.214
```

```
y.215
```

```
y.216
```

```
y.217
```

```
y.218
```

```
y.219
```

```
y.220
```

```
y.221
```

```
y.222
```

```
y.223
```

```
y.224
```

```
y.225
```

```
y.226
```

```
y.227
```

```
y.228
```

```
y.229
```

```
y.230
```

```
y.231
```

```
y.232
```

```
y.233
```

```
y.234
```

```
y.235
```

```
y.236
```

```
y.237
```

```
y.238
```

```
y.239
```

```
y.240
```

```
y.241
```

```
y.242
```

```
y.243
```

```
y.244
```

```
y.245
```

```
y.246
```

```
y.247
```

```
y.248
```

```
y.249
```

```
y.250
```

```
y.251
```

```
y.252
```

```
y.253
```

```
y.254
```

```
y.255
```

```
y.256
```

```
y.257
```

```
y.258
```

```
y.259
```

```
y.260
```

```
y.261
```

```
y.262
```

```
y.263
```

```
y.264
```

```
y.265
```

```
y.266
```

```
y.267
```

```
y.268
```

```
y.269
```

```
y.270
```

```
y.271
```

```
y.272
```

```
y.273
```

```
y.274
```

```
y.275
```

```
y.276
```

```
y.277
```

```
y.278
```

```
y.279
```

```
y.280
```

```
y.281
```

```
y.282
```

```
y.283
```

```
y.284
```

```
y.285
```

```
y.286
```

```
y.287
```

```
y.288
```

```
y.289
```

```
y.290
```

```
y.291
```

```
y.292
```

```
y.293
```

```
y.294
```

```
y.295
```

(d) set of all strings such that 10th symbol from the right end is 1. (binary)

$\Rightarrow (0+1)^* 1 (0+1)^9$

y.y.

$[0-9]^* 1 [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]$
($\{ \text{yylex} \text{ ("valid")}, \cdot \}$)

y.y. \rightarrow main() {
int yywrap();
{
return 0;
3

int main()
{
yyless();
return 0;

output:

101068691745

\rightarrow valid

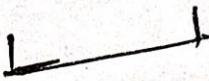
107664867

\rightarrow ..

21345678922

\rightarrow valid

(c) Set of all strings such that every 6 consecutive symbols contain at least two 5's



- 5 5 - -

sol¹⁰ y. {

#include <stdio.h>

#include <math.h>

int value = 0, i = 0, j = 0, flag = 0;

y. {

{ if (yylex[i] == 5)

y. y.

 { for (i = yylex - 2; i >= 0; i--)

 { val = val + (yylex[i] - 8) * pow(2, i); }

 j++,

}

if (value >= 5 == 0)

 flag = 1;

}

[1n] return 0;

y. y.

int main()

{ yylex();

 if (flag == 2)

 { success;

}



3) set of all 4 digit numbers whose sum is 9

DP

Y.Y.

[0-9] [0-9] [0-9] [0-9]

```
int value = 0, i = 0;
for (i = 0; i < 4; i++) {
    value = value + (ytext[i] - 48);
    if (value == 9) {
        printf("Number %s matching the given condition, ytext");
    }
}
```

Q.

Output:

1233

sum = 9

1431

sum = 9

Q.

ZPF1P0220101

b10

608100F01

55P34072P815

3788

Q) Write a program to perform Recursive descent parsing on the following grammar: $S \rightarrow cAd$, $A \rightarrow ab \mid a s$

```

main()
{
    S();
}

S()
{
    if (look-ahead == '$')
    {
        print "success";
    }
}

A()
{
    if (look-ahead == 'c')
    {
        match('c');
        A();
        match('a');
    }
}

A()
{
    if (look-ahead == 'ab')
    {
        match('a');
        match('b');
    }
}

else if (look-ahead == 'a')
{
    match('a');
}

else
{
    return ("invalid");
}

```

```

match(char c)
{
    if (look-ahead == c)
    {
        if (look-ahead == '$')
        {
            print("success");
        }
        else
        {
            print("error");
        }
    }
}

```

~~cad\$~~ (valid)

~~cad\$~~ (valid)

~~cad\$~~ (valid)

~~cad\$~~ (valid)

~~cad\$~~ (valid)

~~cad\$~~ (valid)

char look_ahead;

void match(char c)

{ if (look_ahead == c)

{ look_ahead = getchar(); }
else

printf("Error");
exit(1); }
else

void A()

{ if (look_ahead == 'a')

{ match('a');
if (look_ahead == 'b')
{ match('b'); }
else {

printf("Error\n"),
exit(1); } }
else {

void S()

{ if (look_ahead == 'c')

{ match('c');
A(),
match('d'); }
else {

output:

Enter string: cab

string is accepted

Enter string: cabd

string is accepted

✓ See 261, 2123 : ('b') return
: ('a') return

int main()

{ look_ahead = getchar(); }

if (look_ahead == EOF) {
printf("Success\n"); }
else {

printf("Error\n"); }
return 0; }
else {
printf("Error\n"); }
return 0; }
else {
printf("Error\n"); }
return 0; }

sol 5

y. {

#include <sys/types.h>, <sys/stat.h> etc
external int yytext;

y. }

y.y.

[0-9] + digit = atoi (system); return digit;

[+/-] + (E+F)

[n] return 0;

return yytext[0];

y.y.

int yywrap () {

}

prog. 8:

y. {

#include <ctype.h> // for isdigit() <- standard library

#include <stdio.h>

#include <stdlib.h> // for atoi() standard library

x} 0

y. token digit

y. y.

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

E: E + T { printf (" + "); }, + is operator, T is token, so also tokens

| E - T { printf (" - "); }, - is operator, so also tokens

| T * F { printf (" * "); }, * is operator, so also tokens

| T / F { printf (" / "); }, / is operator, so also tokens

| F

;

F: F * R { printf ("\n"); }

$x \in X$

I disit { printf("%d", &x); }

;

Y.Y.

int main()

{
 printf("Enter infix expr: ");

yy

return 0;

}

void yyerror()

{
 printf("Error in ");
}



$$\begin{aligned} & 2+5*3 \\ & = 25^3*+ \end{aligned}$$

Output:

Enter infix expression:

$4 + 3 * (5 - 9) / 4 ^ 4$

$\Rightarrow 4259 -$

~~23/1/24~~
23/1/24

Q) Write a C program to design lexical analysis to recognize any five 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 < 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::

```

in.c
#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);
        }
    }
}

```

Output

```

/tmp/AWz7FTr3It.o
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 7

1.1 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;}\n.\n{ return yytext[0];}\n%%\nint yywrap()\n{\n}
```

YACC

```
%{\n#include<stdio.h>\n\n#include<stdlib.h>\n\nint yyerror(const char *s);\nint yylex(void);\n%}\n%token num;\n%left '+' '-'\n%left '*' '/'\n%left ')'\n%left '('\n%%\ns:e {printf("Valid expression!\n");\n\nprintf("Result:%d\n", $$); exit(0);\n}\n;\ne:+e { $$=$1+$3;}\n|e:-e { $$=$1-$3;}\n|e'*e { $$=$1*$3;}\n|e/'e { $$=$1/$3;}\n|'('e')' { $$=$2;}\n|num { $$=$1;}\n;\n%%
```

```

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

```

1.2 Write a program in YACC to recognize strings of the form {(a^n)b , n>=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 S B NL {printf("Parsed using the rule (a^n)b, n>=5.\nValid String!\n");}

;

%%

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

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

```

Output:

```

Enter a string!
abc
Invalid String!

```

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

1.3 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
```

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=%d\n",iden,var_cnt-1);}

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

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

|T {$$=$1;

;

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

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

|F {$$=$1;

;

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

|P {$$ = $1;

;

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

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

;

% %

int main()
{
var_cnt=0;
printf("Enter an expression:\n"); yyparse();

```

```
return 0;  
}  
  
int yyerror(char *s)  
{  
    printf("Invalid expression!"); return 0;  
}
```

Output:

```
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
```