VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

COMPILER DESIGN

Submitted by

Likhith G S(1BM21CS096)

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 **Likhith G S(1BM21CS096)** , 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	Dr. Jyothi Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Likhith G S (1BM21CS096), 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

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.

Lab 1

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

Code:

```
① Write a lex program to identify - datatype, int,
  char, float & variables.

%option noyywrap
%{
    #include<stdio.h>
%}
%%
[0-9]* {printf("\n %s is Integer", yytext); }
[a-zA-Z]* {printf("\n %s is character", yytext); }
[-+]? [0-9]* [.][0-9]+ {printf("\n %s is float", yytext
[a-zA-Z]+ {printf("\n %s is variable", yytext);
%%
void main()
{
    yylex();
}
```

(No space for R.E)

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) WALP to count the number of vowels & consonants in a given string.

```
int vowel = 0;
int consonant = 0;
%%
[aeiouAEIOU] { vowel++; }
[a-z A-Z] { consonant++; }
"\n" { printf("Number of vowels is %d and
        Number of consonants is %d", vowel,
        consonant); }
%%
```

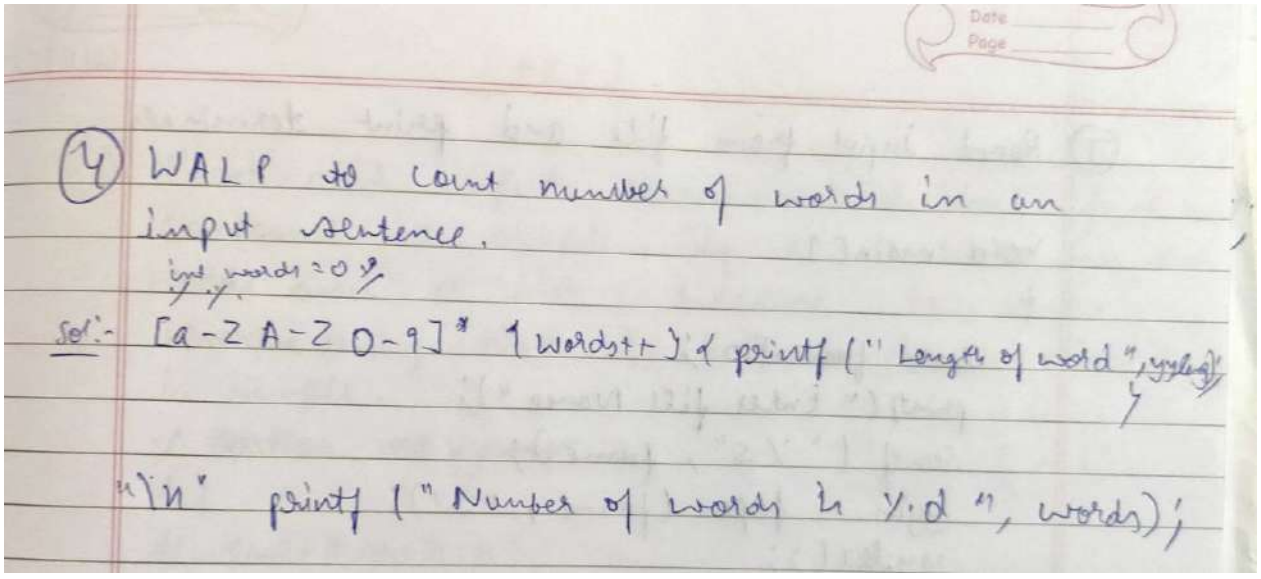
Output

```
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
^C
```

Lab 2

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

Code:



(4) WALP to count number of words in an input sentence.

int words = 0;

Sol:- [a-z A-Z 0-9]* {words++} printf("Length of word", yyleng);

"\n" printf("Number of words is %d", words);

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:

```
(3) WALP to identify alphabets as characters & numbers as digits

[a-z A-Z]* {printf("\n%s is stream of characters", yytext); }

[-+]? [0-9]* { printf("\n%s is a digit", yytext);}
```

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.

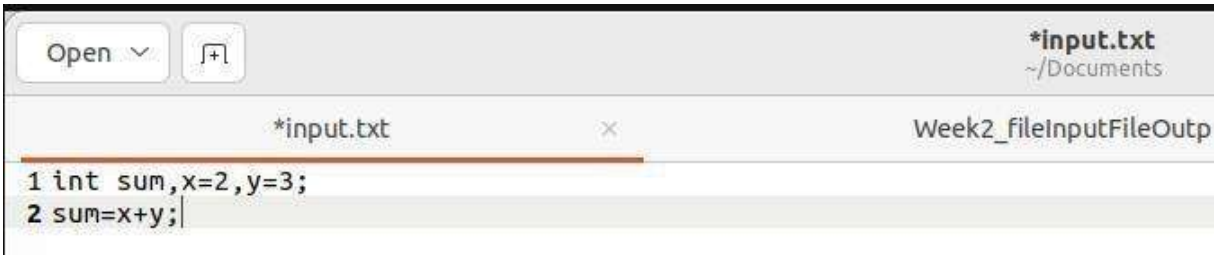
Code:

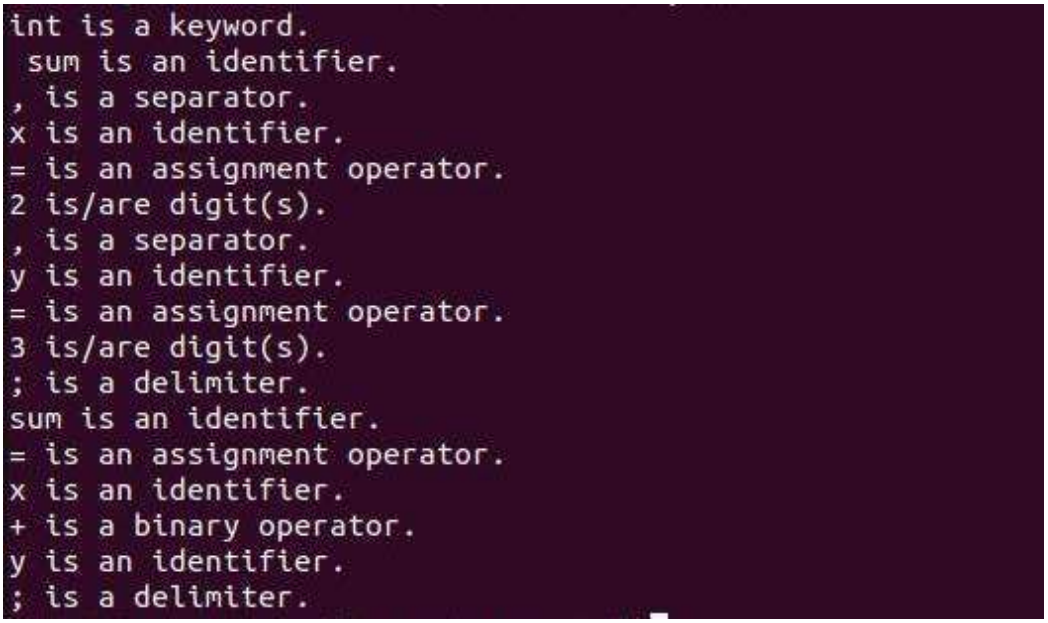
Date
Page

(7) Read input from file and print terminals.

```
void main()
{
    char fname[10];
    printf(" Enter file Name ");
    scanf(" %s", fname);
    yyin = fopen(fname, "r");
    yylex();
    fclose(yyin);
}
```

Output


Open
*input.txt
~/Documents
*input.txt
Week2_fileInputFileOutp
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 but output stored in another file. Ask for output file name.

```
yyin = fopen(fname, 'r');
yyout = fopen(opname, "w");
yylex();
fclose(yyin);
fclose(yyout);
return 0;

int main() {
    char fname[20];
    extern FILE *yyin;
    printf(" Enter the Input file \n");
    scanf("%s", fname);
    yyin = fopen("fname", "r");
    yylex();
    return 0;
}
```

Output

Open

*input.txt
~/Documents

*input.txt

Week2_fileInputFileOutp

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

```
Printed in output.txt
```

Open

*output.txt
~/Documents

input.txt

Week2_fileInputFileOut

```
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

2) Write a program in LEX to recognize floating point numbers.

```
%%
[+-]?[0-9]*[.][0-9]+ { printf("Floating point numbers"); }
[+-]?[0-9]+ { printf(" Not floating"); }

%%
```

Output

```
Enter a Number
34
Not floating
3.4
floating
```

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

```
%option noyywrap
%{
#include <stdio.h>
int flag = 0;
%}
%%
and | or | but | because | if | then | nevertheless { flag=1; }
"\n" { if (flag==1)
            printf (" Yes, compound");
       else
            printf (" simple");
     }
%%
int main()
{ printf ("Enter a string");
  yylex();
  return 0;
}
```

Output

```
Enter a string:
Hi hello.
=> simple.
we go out and in
=> compound.
```

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) WALP to check '.' or ! (or) ? is present at the end of sentence

```
%%
[a-z A-Z] [.?!] {flag = 1;}
\n {return 0;}
%%
```

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

5) WALP to check, if it begin with the article or not.

```
%%
[A|a|AN|An|THE|The] [" "] {flag=1;}
\n {return 0;}
%%
```

Output :-

An Dog → Yes,

Dog → No

27/11/23

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

```
Count Number of comment lines.

%{
#include <stdio.h>
int c=0;
%}
%%
"\n*"[n*]*\* + ([^/*][n*]*\*+)
        * v{ c++;}

"//" * { c++; }
. ECHO;
%%

void main()
{
   yyin = fopen("input.txt","r");
   yyout = fopen("output.txt","w");
   yylex();
   printf(" The Number of comment lines 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
```

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

Code

```
4) WALP to check signed / unsigned Integer.

%.%.
[+-]?[0-9]+ { printf("%s is signed Integer",
                    yytext); }
[0-9]+ { printf("%s is unsigned Integer",
                    yytext); }
%.%.
```

Output :-

34 → Unsigned Integer

-34 → Signed Integer.

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

1) Write a LEX program that copies a file, replacing each non empty sequence of white spaces by a single blank.

Sol:-

```
%option noyywrap.
%{
    #include<stdio.h>
    #include<string.h>
    #include<stdlib.h>
%}
%%
[\n]   {fprintf(yyout, "%s\n", string); string[0]='\0';
[ ]*   {fprintf(yyout, "%s", string); string[0]='\0';
        fprintf(yyout, "%s", " "); }

. strcat(string, yytext);
<<EOF>>  {fprintf(yyout, "%s", string); return 0;}
%%

int main()
{
    extern FILE *yyin, *yyout;
    char filename[100];
    printf(" Enter file name : ");
    scanf("%s", filename);
    yyin = fopen(filename, 'r');
    if(yyin == NULL){
      printf(" can't open file %s\n", filename);
      exit(0);
    }
    printf("Enter the name of file to open:");
    scanf("%s", filename);
```

```
yyout = fopen(filename, 'w');
if (yyout == NULL){
    printf(" Can't open file %s\n", filename);
    exit(1);
}
yylex();
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 strings ending in 00.

Code

```
2)
a,b) [a-z A-Z 0-9]* [0][0]$ { printf("Yes 00 at end \n"); }
```

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

```
[a-z A-Z 0-9]* [222] [a-z A-Z 0-9]*
    printf("Yes, 222 present \n"); }
```

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

2)

d)

```
#include <math.h>
1[01]* { for (i = yyleng - 1; i >= 0; i--).
{
    value += (yytext[i] - '0') * pow(2, j);
    j++;
}
if (value % 5 == 0)
{
    printf("Success \n");
}
else
    printf("Fail \n");
}
```

4/12/2023

Output

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

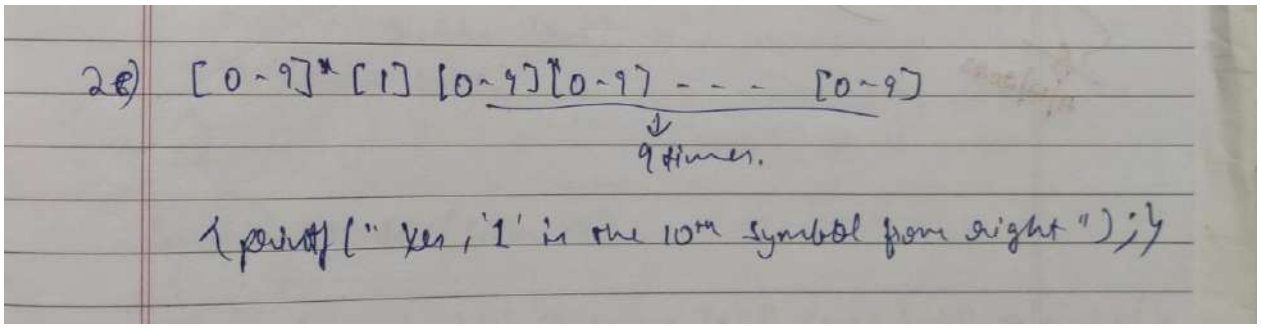
```
Enter a string:
1010
Decimal representation:10
Congruent to modulo 5.
```

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

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

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

1. Code



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

```
2f)
[0-9][0-9][0-9][0-9] { for ( i= yylenq-1; i>=0;
                                        i--)
{ value += (yytext[i] - '0');
}
  if (value == 9).
      printf (" sum is 9 \n");
else
      printf (" Fail");
}
```

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

```
digits [0 - 9].
%%
{digits} {digits} {digits} {digits}
{
    for (i=0; i< yyleng-1; i++)
    {
        if (yytext[i] > yytext[i+1])
            flag=0;
    }
}
[\n] return 0;

output
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

18/12/22

(1) Write a C/C++ program to design Lexical analyzer to recognize any five Keywords, identifiers, numbers, operators and punctuations.

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

bool isDelimeter (char ch).
{
    if ( ch == ' ' || ch == 'x' || ch == 'y' ||
        ch == '+' || ch == '-' || ch == '*' || ch == '/' )
        return true;
    return false;
}

bool isOperator (char ch)
{
    if ( ch == '+' || ch == '-' || ch == '*' || ch == '/' ||
        ch == '>' || ch == '<' ).
    return true;
    return false;
}

bool validIdentifier ( char *str )
{
    if ( str[0] == '0' || str[0] == '1' || ... ||
        str[0] == '9' )
        return false;
    return true;
}
```

```
bool isKeyword ( char * str)
{
    if ( ! strcmp ( str, "if") ||
! strcmp ( str, "else") || ! strcmp ( str, "while"
  --- || ! strcmp ( str, "char") )
    return true;
return false;
}


bool isInteger ( char * str).
{
    int len = strlen(str);
    if ( len == 0) return false;
    for ( i=0; i<len; i++) {
        if ( str[i] != '0' && str[i] != '1' ||
            -- str[i] != '9' )
            return false;
    }
    return true;
}


void parse ( char * str ).
{
    int l=0, r=0;
    int len = strlen ( str );

    while ( r <= len && l <= r) {
        if ( isDelimeter ( str[r] ) == false )
            r++;
```

```
if ( isKeyword (substr) == true )
        printf (" %s is keyword \n", substr );
else if ( isInteger(substr) == true)
        printf(" %s is Integer \n", substr);
else if ( validIdentifier (substr ) == false)
        printf (" %s is valid Identifier ", substr);

}
}
return;
}
```

~~Output~~ void main( )
{
char str[100] = "int a = b + c;",
parse (str);
}

Output :-

'int' is a keyword.
'a' is a valid identifier
'=' is a operator
'b' is a valid identifier
'+' is an operator
'c' is not a valid identifier

10/10

18/12/2023

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

124

① Write a Program to perform Recursive Descent Parsing on the following grammar.

S → cAd , A → ab/a.

Sol:-

```
#include<stdio.h>
#include<stdlib.h>
void A();
char input[100];
int ind=0;

void match (char expected)
{
    if (input[ind] == expected)
        ind++;
}

void S()
{
    match ('c');
    A();
    match ('d');
}

void A()
{
    if (input[ind] == 'a')
    {
        printf (" Parsing Success for...\n");
        match ('b');
    }
```

```
else {
    printf(" Parsing failed %d \n", ind);
    exit(1);
}
}

int main() {
    printf(" Enter the input string! \n");
    scanf("%s", input);

    S();

    if (input[ind] == '$')
        printf(" Parsing successful \n");
    else
        printf(" Parsing failed. Extra characters found \n");

    return 0;
}
```

Output

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

```
Enter a string:
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 +, -, * and /.

Code

② Design a suitable grammar for evaluation of arithmetic expression having + and - operators.

+ have least priority & it is left associative
- has higher priority & is right associative

Sol: l.l file

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

l.y file

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

%token NUM
%left '+'
%right '-'

%%
```

```
expr: e { printf("valid expression\n"); printf("Result
    : %d\n", $$); return 0; }

e: e'+'e.      { $$ = $1 + $3; }
| e'-'e        { $$ = $1 - $3; }
| NUM          { $$ = $1; }

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

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

* <u>To execute:-</u>

- lex p.l
- yacc -d p.y
- gcc lex.yy.c y.tab.c
- ./a.out

Sun 8/1/24

<u>OUTPUT:-</u>

Enter. Arthimetic expression:
~~3+4-24~~ 5+6-3-6
Valid expression.
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>=5}.

Code

29/1/24

LAB - 7

String Matching

Lex file :-

```
%{
#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 file :-

```
%{
#include <stdio.h>
int yyerror (char *s);
int yylex (void);
%}
%token A
%token B
%token NL
%%
Start: A A A A S B NL { printf("Parsed valid
 the rule (a^n)b}n"); }
;
```

```
S: S A
|
;
%%
void main()
{
  printf("Enter a string\n");
  yyparse();
}
int yyerror(char *s)
{
  printf("Invalid string\n");
  return 0;
}
```

Output:-

Enter a string
aaaaaaab

Parsed using the rule (a^n)b, n>=5

Valid string

ab.
Invalid string.

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

27/1/23

Syntax Tree

Lex file :-

```
%{
#include <stdio.h>
#include <stdlib.h>
# include "y.tab.h"
extern int yylval;
%}
[0-9]+ { yylval = atoi (yytext); return digit; }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
    return 1;
}
```

yacc file :-

```
%{
#include <math.h>
# include <string.h>
int yyerror (char *s);
int yylex (void);

struct tree_node {
  char val[10];
  int lc, rc;
};
```

```
int mknode (int lc, int rc, char * val);
%}

%token digit
%%
S: E { my-print tree (S1); }
;
E: E '+' T { $$ = mknode ( $1, $3, "+"); }
| T { $$ = $1; }
;
T: T '*' F { $$ = mknode ( $1, $3, '*'); }
| F = { $$ = $1; }.
%%

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

int yyerror (char * s).
{
  printf (" NIW error \n");
  return 0;
}

void my-print-tree (int c-id)
{
  if ( c-id == -1) return;
  if ( syn-tree ( c-id ) . lc == -1 && syn-tree
        [c-id]. rc == -1)
    printf ( " Digit node -> Index : %d , value %s
        c-id , syn-tree[c-id]. val);
```

```
else
  my_print_tree ( Syn_tree[c_id] . lc );
  my_print_tree ( Syn_tree[c_id] . rc );
}
```

Output :-

```
lex  Inf_to_post.l
yacc Inf_to_post.y
gcc lex.yy.c y.tab.c
```

~~Enter an infix expression.~~
~~2 + 3 * 8 / 4 ^ 3 - 3~~

~~2 3 8 * 4 3 ^ / + 3 -~~

Enter an expression.
2 * 3 + 5 * 4.
Operator Node → Index : 6
value : + , LeftChild Ind : 2 , RightChild Ind : 5

Operator Node → Ind : 2, val = * , LeftChild Ind : 0,
RightChild Ind : 1.
Digit Node → Ind : 0 , value : 2.
Digit Node → Ind : 1 , value : 3

Operator Node → Ind : 5 , value : 3
Digit Node → Ind : 4 , value : 4.

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:

Week 8

Infix to Postfix

Lex file:

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

Yacc file:-

```
%{
#include <stdio.h>
int yyerror (const char *s);
int yylex(void);
%}
%token num
%left '+' '-'
%left '*' '/'
%left ')'
%left '('
%right '^'
%%
```

```
S : e { printf ("\n"); }
;
e : e '+' t { printf ("+"); }
| e '-' t { printf ("-"); }
| t
;
t : t '*' h { printf ("*"); }
| h
;
f : '(' e ')'
| num { printf ("%d", $1); }
;
| f
;
void main ()
{
  printf ("Enter infix exp:\n");
  yyparse();
}

int yyerror (char *s)
{
  printf ("Invalid infix exp\n");
  return 0;
}
```

Output:-

Enter an infix expression

2+3*8/4^3-3

238*43^/+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:

29/1/24

Week - 9

Three address code

Lex file :-

```
%{
#include <stdio.h>
#include "y.tab.h"
extern int yylval;
extern char ind[20];
%}
%%
d[0-9]+
a[a-z A-Z]+
%%
{d} { yylval = atoi(yytext); return digit; }
[\t] { ; }
\n return 0;
return yytext[0];
%%
int yywrap() { return 1; }
```

yacc file :-

```
%{
#include<math.h>
#include<stdio.h>
int yyerror(char *s);
int varcnt = 0;
char ind[20];
%}

%token id
%token digit
%%
```

```
S: id '=' E { printf("%s = %d %d\n", idar,
                varcnt - 1); }
E : E '+' T { $$ = varcnt; varcnt++; }
  | E '-' T { $$ = varcnt; varcnt++; printf(
      "t%d = t%d - t%d; \n", $$, $1, $3);
  | T { $$ = $1; }
  ;
T : T '*' F { $$ = var_cnt; var_cnt++; } ;
  | T '/' F { $$ = var_cnt; var_cnt++; printf("
      t%d = t%d / t%d; \n", $$, $1, $3); }
  | F { $$ = $1; }
  ;

F : '(' 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;

29/1/2024

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
```