Programme outcomes (POs)

a. An ability to apply the fundamental knowledge of mathematics, computing, science, and engineering to solve Computer Science and Engineering problems**.**
b. An ability to design and conduct engineering experiments as well as to analyze and interpret data with rubrics.
c. An ability to design and construct a hardware and software system, component, or process to meet desired needs, within realistic constraints with core instruction and state-of-the art knowledge.
d. An ability to function on multi – disciplinary teams**.**
e. An ability to identify, formulate and solve engineering problems in the Computer Science and Engineering field.
f. An understanding of professional, social and ethical responsibility with informed citizenship.
g. An ability to apply the broad education necessary to understand the impact of engineering solutions in a global, economic, and societal context.
h. An ability to recognize the need for and to engage in life – long learning.
i. An ability to apply computational skills and knowledge to maintain environmental sustainability.
j. An ability to use computer programming skills and modern engineering tools in networking and security necessary for engineering practice.
k. An Ability to communicate effectively in both verbal and written usage.
l. An Ability to understand and implement inter disciplinary concepts for project management and finance.

Programme Specific Outcome

m. An ability to get an employment in Computer Science and Engineering field and related software industries and to participate & succeed in competitive examinations like GRE,GATE,TOEFL,GMAT etc.

Sl. No.	List of Experiments	Page No.
1.	1. Write a program to search for a given pattern in a set of files. It should support regularexpressions. It should work similar to grep and fgrep of Linux environment.	
2.	2. Write programs for DFA, NFA.	
3.	3. Consider the following regular expressions: a) (0 + 1)* 1(0+1)(0+1) b) (ab*c + (def)+ + a*d+e)+ c) ((a + b)*(c + d)*)+ + ab*c*d Write separate programs for recognizing the strings generated by each of the regular expressionsmentioned above (Using FA).	
4.	4. Given a text-file which contains some regular expressions, with only one RE in each line of thefile. Write a program which accepts a string from the user and reports which regular expressionaccepts that string. If no RE from the file accepts the string, then report that no RE ismatched.	
5.	5. Design a PDA for any given CNF. Simulate the processing of a string using the PDA and showthe parse tree.	
6.	6. Design a Lexical analyzer for identifying different types of tokens used in C language	
7.	7. Simulate a simple desktop calculator using any lexical analyzer generator tool (LEX or FLEX).	
8.	8. Program to recognize the identifiers, if and switch statements of C using a lexical analyzer generator tool.	
9.	9. Consider the following grammar: S --> ABC A--> abA | ab B--> b | BC C--> c | cC Design any shift reduced parser which accepts a string and tells whether the string is accepted by above grammar or not	
10.	10. Design a YACC program that reads a C program from input file and identify all valid C identifiers and for loop statements.	
11.	11. Program to eliminate left recursion and left factoring from a given CFG.	
12.	12. YACC program that reads the input expression and convert it to post fix expression.	
13.	13. YACC program that finds C variable declarations in C source file and save them into the symbol table, which is organized using binary search tree.	
14.	14. YACC program that reads the C statements from an input file and onverts them into quadruplethree address intermediate code.	

Note 1:

A simple language written in this language is

```
{int a[3],t1,t2;
T1=2;
A[0]=1;a[1]=2;a[t]=3;
T2=-( a[2]+t1*6)/(a[2]-t1);
If t2>5then
Print(t2)
Else{
Int t3;
T3=99;
T2=25;
Print(-t1+t2*t3);/*this is a comment on 2 lines*/
}endif
}
```

Comments(zero or more characters enclosed between the standard C/JAVA Style comment brackets/*…*/)can be inserted .The language has rudimentary support for1-dimenstional array,the declaration int a[3] declares an array of three elements,referenced as a[0],a[1] and a[2].Note also you should worry about the scopping of names.

Note 2:

Consider the following mini language, a simple procedural high –level language, only operating on integer data, with a syntax looking vaguely like a simple C crossed with pascal. The syntax of the language is defined by the following grammar.

```
<program>::=<block>
<block>::={<variable definition><slist>}
|{<slist>}
<variabledefinition>::=int <vardeflist>
<vardec>::=<identifier>|<identifier>[<constant>]
<slist>::=<statement>|<statement>;<slist>
<statement>::=<assignment>|<ifstament>|<whilestatement>
|<block>|<printstament>|<empty>
<assignment>::=<identifier>=<expression>
|<identifier>[<expression>]=<expression>
<if statement>::=if<bexpression>then<slist>else<slist>endif
|if<bexpression>then<slisi>endif
<whilestatement>::=while<bexpreession>do<slisi>enddo
<printstatement>:;=print(<expression>)
<expression>::=<expression>::=<expression><addingop><term>|<term>|<addingop>
<term>
<bexprssion>::=<expression><relop><expression>
<relop>::=<|<=|==|>=|>|!=
<addingop>::=+|-
<term>::=<term><multop><factor>|<factor>
<Multop>::=*|/
<factor>::=<constant>|<identifier>|<identifier>[<expression>]
|(<expression>)
<constant>::=<digit>|<digit><constant>
<identifier>::=<identifier><letter or digit>|<letter>
<letter or digit>::=<letter>|<digit>
<letter>:;=a|b|c|d|e|f|g|h|I|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit>::=0|1|2|3|4|5|^|7|8|9
<empty>::=has the obvious meaning
```

Exp. No.	**Experiment**	**Program Outcomes Attained**	**Program Specific Outcomes Attained**
1.	1. Write a program to search for a given pattern in a set of files. It should support regular expressions. It should work similar to grep and fgrep of Linux environment.	a,b,e,j	m
2.	2. Write programs for DFA, NFA.	a,b,e,j	m
3.	3. Consider the following regular expressions: a) (0 + 1)* 1(0+1)(0+1) b) (ab*c + (def)+ + a*d+e)+ c) ((a + b)*(c + d)*)+ + ab*c*d Write separate programs for recognizing the strings generated by each of the regular expressionsmentioned above (Using FA).	a,b,e,j	m
4.	4. Given a text-file which contains some regular expressions, with only one RE in each line of thefile. Write a program which accepts a string from the user and reports which regular expressionaccepts that string. If no RE from the file accepts the string, then report that no RE ismatched.	a,b,e,j	m
5.	5. Design a PDA for any given CNF. Simulate the processing of a string using the PDA and showthe parse tree.	a,b,e,j	m
6.	6. Design a Lexical analyzer for identifying different types of tokens used in C language	a,b,e,j	
7.	7. Simulate a simple desktop calculator using any lexical analyzer generator tool (LEX or FLEX).	a,b,e,j	
8.	8. Program to recognize the identifiers, if and switch statements of C using a lexical analyzer generator tool.	a,b,e,j	m
9.	9. Consider the following grammar: S --> ABC A--> abA | ab B--> b | BC C--> c | cC Design any shift reduced parser which accepts a string and tells whether the string is accepted by above grammar or not	a,b,e,j	m
10.	10. Design a YACC program that reads a C program from input file and identify all valid C identifiers and for loop statements.	a,b,e,j	
11.	11. Program to eliminate left recursion and left factoring from a given CFG.	a,b,e,j	
12.	12. YACC program that reads the input expression and convert it to post fix expression.	a,b,e,j	
13.	13. YACC program that finds C variable declarations in C source file and save them into the symbol table, which is organized using binary search tree.	a,b,e,j	
14.	14. YACC program that reads the C statements from an input file and onverts them into quadruplethree address intermediate code.	a,b,e,j	

COMPILER DESIGN LABORATORY

OBJECTIVE:

This laboratory course is intended to make the students experiment on the basic techniques of compiler construction and tools that can used to perform syntax-directed translation of a high-level programming language into an executable code. Students will design and implement language processors in C by using tools to automate parts of the implementation process. This will provide deeper insights into the more advanced semantics aspects of programming languages, code generation, machine independent optimizations, dynamic memory allocation, and object orientation.

OUTCOMES:

Upon the completion of Compiler Design practical course, the student will be able to:

1.Develop compiler tools
2.Design simple compiler

EXPERIMENT-1

1.1 OBJECTIVE:

Write a program to search for a given pattern in a set of files. It should support regular expression and should work similar to grep and fgrep of Linux environment.

1.2 RESOURCE: Turbo **C**

1.3 PROGRAM LOGIC

1.Enter the text
2.Enter the text to search
3.If pattrenlenght=textlength,if pattern =text,then position of text is found
4.else
5.text is not found.

1.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

1.5 PROGRAM:

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

int match(char [], char []);

int main() {
  char a[100], b[100];
  int position;

  printf("Enter some text\n");
  gets(a);

  printf("Enter a string to find\n");
  gets(b);

  position = match(a, b);

  if(position != -1) {
    printf("Found at location %d\n", position + 1);
  }
  else {
    printf("Not found.\n");
  }

  return 0;
}

int match(char text[], char pattern[]) {
  int c, d, e, text_length, pattern_length, position = -1;

  text_length    = strlen(text);
  pattern_length = strlen(pattern);

  if (pattern_length > text_length) {
    return -1;
  }

  for (c = 0; c <= text_length - pattern_length; c++) {
    position = e = c;

    for (d = 0; d < pattern_length; d++) {
      if (pattern[d] == text[e]) {
        e++;
      }
      else {
        break;
      }
    }
    if (d == pattern_length) {
      return position;
    }
  }

  return -1;
}
```

1.5 INPUT & OUTPUT:

Input

```
Enter some text
Hai hello
Enter a string to find
 Hai
```

OUTPUT:

```
Found at location 00002x0
```

EXPERIMENT-2(a)

2.1 OBJECTIVE:
Write programs for DFA, NFA**.**

2.2 RESOURCE: Turbo C

2.3 PROGRAM LOGIC:

string x

- a DFA with start state, so . . .
- a set of accepting state's F.

- The answer 'yes' if D accepts x; 'no' otherwise.

The function move (S, C) gives a new state from state s on input character C.

The function 'nextchar' returns the next character in the string.

```
Initialization:
        S := $S_0$
      C := nextchar;

while not end-of-file do
        S := move (S, C)
        C := nextchar;

If S is in F then
        return "yes"

else
    return "No".
```

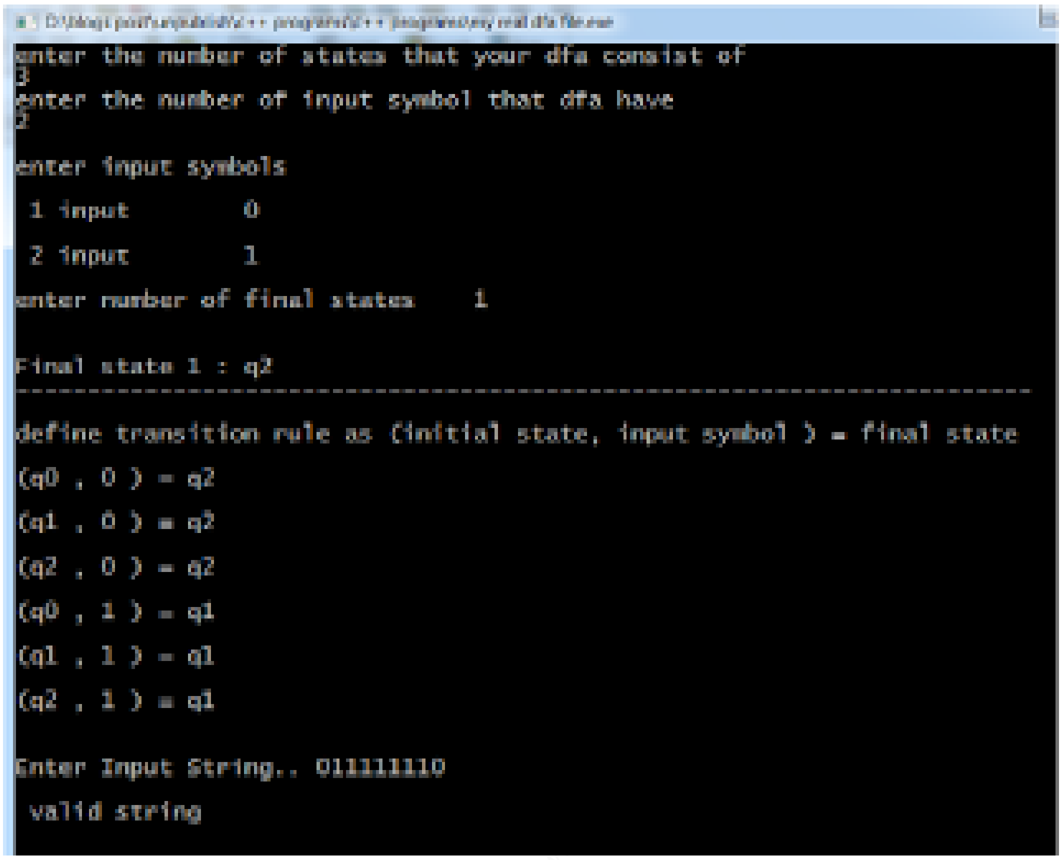
2.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

2.5 PROGRAM:

```
#include<stdio.h>

#include<conio.h>


int ninputs;


int check(char,int );          //function declaration
int dfa[10][10];
char c[10], string[10];
int main()
{
    int nstates, nfinals;
    int f[10];
    int i,j,s=0,final=0;
    printf("enter the number of states that your dfa consist of \n");
    scanf("%d",&nstates);
    printf("enter the number of input symbol that dfa have \n");
    scanf("%d",&ninputs);
    printf("\nenter input symbols\t");
    for(i=0; i<ninputs; i++)
    {
     printf("\n\n %d input\t", i+1);
     printf("%c",c[i]=getch());
    }
    printf("\n\nenter number of final states\t");
    scanf("%d",&nfinals);

    for(i=0;i<nfinals;i++)
    {
      printf("\n\nFinal state %d : q",i+1);
      scanf("%d",&f[i]);
    }

     printf("-----------------------------------------------------------------------");
     printf("\n\ndefine transition rule as (initial state, input symbol ) = final state\n");
     for(i=0; i<ninputs; i++)
     {
            for(j=0; j<nstates; j++)
            {
                    printf("\n(q%d , %c ) = q",j,c[i]);
                    scanf("%d",&dfa[i][j]);
            }
     }
         do
     {
     i=0;
     printf("\n\nEnter Input String.. ");
     scanf("%s",string);
      while(string[i]!='\0')
        if((s=check(string[i++],s))<0)
        break;
        for(i=0 ;i<nfinals ;i++)
        if(f[i] ==s )
        final=1;
           if(final==1)
        printf("\n valid string");
        else
        printf("invalid string");
       getch();

    printf("\nDo you want to continue.?  \n(y/n) ");
     }
     while(getch()=='y');

    getch();
}
int check(char b,int d)
{
    int j;
    for(j=0; j<ninputs; j++)
    if(b==c[j])
    return(dfa[d][j]);
    return -1;
}
```

2.6 INPUT & OUTPUT:



```
enter the number of states that your dfa consist of
3
enter the number of input symbol that dfa have
2

enter input symbols
 1 input        0

 2 input        1

enter number of final states    1

Final state 1 : q2
--------------------------------------------------------------------------

define transition rule as (initial state, input symbol ) = final state
(q0 , 0 ) = q2
(q1 , 0 ) = q2
(q2 , 0 ) = q2
(q0 , 1 ) = q1
(q1 , 1 ) = q1
(q2 , 1 ) = q1

Enter Input String.. 011111110
 valid string
```

EXPERIMENT-2(b)

2.1 OBJECTIVE:

Program for NFA

2.2 RESOURCE: Turbo C

2.3 PROGRAM LOGIC:

Input: an NFA (transition table) and a string x (terminated by eof).

output : “yes” if accepted, “no” otherwise.

```
S = e-closure({s0});

a = nextchar;

while a != eof do begin

    S = e-closure(move(S, a));

    a := next char;

end

if (intersect (S, F) != empty) then return “yes”

else return “no”
```

2.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

2.5 PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
struct current{int first,last;}stat[15];
int l,j,change,n=0,i=0,state=1,x,y,start,final;
char store,*input1,input[15];
clrscr();
printf("\n\n****ENTER THE REGULAR EXPRESSION****\n\n");
scanf("%s",input1);/*ex inputs:1.(a*) 2.(a|b) 3.(a.b) 4.((a|b).(a*))*/
for(i=0;i<10;i++)
input[i]=NULL;
l=strlen(input1);
a:
for(i=0;input1[i]!=')';i++);
for(j=i;input1[j]!='(';j--);
for(x=j+1;x<i;x++)
if(isalpha(input1[x]))
input[n++]=input1[x];
else if(input1[x]!='0')
store=input1[x];
input[n++]=store;
for(x=j;x<=i;x++)
input1[x]='0';
```

```
if(input1[0]!='0')
goto a;
printf("\n\n\tFROM\tTO\tINPUT\n\n");
i=0;
while(input[i]!='\0')
{
if(isalpha(input[i]))
{
stat[i].first=state++;
stat[i].last=state++;
printf("\n\t%d\t%d\t%c",stat[i].first,stat[i].last,input[i]);
}
else
{
change=0;
switch(input[i])
{
case'|':
stat[i].first=state++;
stat[i].last=state++;
x=i-2;
y=i-1;
if(!isalpha(input[y]))
{
b:
switch(input[y])
{
case'*':if(!isalpha(input[y-1]))
{
y=y-1;
goto b;
}
else
x=y-2;
break;
case'|':x=y-3;
break;
case '.':x=y-3;break;
}
change=1;
}
if(!isalpha(input[y]&&change==0))
c:switch(input[x])
{
case '*':
if(!isalpha(input[x-1]))
{x=x-1;goto c;
}
else x=x-2;
break;
case'|':x=x-2;
break;
case '.':x=x-3;
break;
}
printf("\n\t%d\t%d\tE",stat[i].first,stat[x].first);
printf("\n\t%d\t%d\tE",stat[x].last,stat[i].last);
printf("\n\t%d\t%d\tE",stat[i].first,stat[i-1].first);
printf("\n\t%d\t%d\tE",stat[i-1].last,stat[i].last);
start=stat[i].first;
final=stat[i].last;
break;
case'.':
x=i-2;
y=i-1;
if(!isalpha(input[y]))
{
d:
switch(input[y])
{
case'*':if(!isalpha(input[y-1]))
{
y=y-1;
goto d;
}
else
x=y-2;
break;
case'|':x=y-3;
break;
case '.':x=y-3;
break;
}
change=1;
}
if(!isalpha(input[y]&&change==0))
e:switch(input[x])
{
case'*':
if(!isalpha(input[x-1]))
{
x=x-1;
goto e;
}
else x=x-2;
break;
case'|':x=x-3;
break;
case'.':x=x-3;
break;
}
stat[i].last=stat[y].last;
stat[i].first=stat[x].first;
printf("\n\t%d\t%d\tE",stat[x].last,stat[i-1].first);
start=stat[x].first;
final=stat[i-1].last;
break;
case'*':
stat[i].first=state++;
stat[i].last=state++;
printf("\n\t%d\t%d\tE",stat[i].first,stat[i-1].first);
printf("\n\t%d\t%d\tE",stat[i].first,stat[i].last);
printf("\n\t%d\t%d\tE",stat[i-1].last,stat[i-1].first);
printf("\n\t%d\t%d\tE",stat[i-1].last,stat[i].last);
start=stat[i].first;
final=stat[i].last;
break;
}}
```

```
i++;
}
printf("\n the starting state is %d",start);
printf("\n the final state is %d",final);
getch();
}
```

2.6 INPUT & OUTPUT:

INPUT :

```
 ****ENTER THE REGULAR EXPRESSION****

((a|b)*
```

OUTPUT:

```
****NFA FOR THE GIVEN REGULAR EXPRESSION****
    FROM        TO      INPUT
      1          2       a
      3          4       b
      5          1       E
      2          6       E
      5          3       E
      4          6       E
      7          5       E
      7          8       E
      6          5       E
      6          8       E
```

EXPERIMENT-3(a)

3.1 OBJECTIVE:

Consider the following regular expressions:
a)(0+1)*1(0+1)(0+1)

3.2 RESOURCE: Turbo C

3.3 PROGRAM LOGIC:

1.Enter the regular expression
2.If the combination is 01 with three 1s then
3.String is valid
4.else not valid

3.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

3.5 PROGRAM:

3.6 INPUT & OUTPUT:

EXPERIMENT-3(b)

3.1 OBJECTIVE:

Program to recognize the string generated by the regular expression (**ab*c+(def)$^+$+a*d$^+$e)+**

3.2 RESOURCE: Turbo C

3.3 PROGRAM LOGIC:

1.Enter the regular expression
2.If the combination is aba with one or more than def then
3.String is valid
4.Else string is not valid

3.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

3.5 PROGRAM:

3.6 INPUT & OUTPUT:

EXPERIMENT-3(c)

3.1 OBJECTIVE:

Program to recognize the string generated by the regular expression **$((a+b)^*(c+d)^*)^+ + ab^*c^*d$**

c) $((a+b)^*(c+d)^*)^+ + ab^*c^*d$

3.2 RESOURCE: Turbo C

3.3 PROGRAM LOGIC:

1.Enter the regular expression
2.If the combination is abcd then
3.String is valid
4.Else string is not valid

3.4 PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

3.5 PROGRAM:

3.6 INPUT & OUTPUT:

EXPERIMENT-4

4.1 OBJECTIVE:

Given a text file which contains some regular expressions, with only one RE in each line of file. Write a program which accepts a string from the user and reports which expression accepts the string. If no RE from the file accepts the string, then report that none is matched**.**

4.2 RESOURCE: Turbo C

4.3 PROGRAM LOGIC:

1.Enter the regular expression
2.Enter the text
3. If reg[0] == '*' then consider the regular expression as invalid
4.Else if the regular expression starts with Alphabet and index is equal,then the regular expression accepts the string.

4.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

4.5 PROGRAM:

```
#include <stdio.h>
#include <string.h>
#define MATCH printf("\nThe Text Matches The Regular Expression");
#define NOTMATCH printf("\nThe Text Doesn't match the Regular Expression");

char reg[20], text[20];

int main()
{
    int i, rlen, tlen, f = 0;
    char ans;

    do {
            printf("\nEnter the Regular Expression\n");
            scanf(" %[^\n]s", reg);
            for (rlen = 0; reg[rlen] != '\0';rlen++);
            printf("\nEnter the text\n");
            scanf(" %[^\n]s", text);
            for (tlen = 0;text[tlen] != '\0' ; tlen++);
            if (reg[0] == '*')
            {
                printf("\nInvalid regular expression");
            }
            /*
             *If the regular expression starts with Alphabet
             */
            if ((reg[0] >= 65 && reg[0] <= 90) || (reg[0] >= 97 && reg[0] <=122))
            {
                if (reg[0] == text [0])
                {
                    switch (reg[1])
                    {
                    case '.' :
                        switch (reg[2])
                        {
                        case '*':
                            if (tlen != 1)
                            {
                                if (reg[3] == text[tlen-1])
                                {
                                    MATCH;
                                }
                                else
                                {
                                    NOTMATCH;
                                }
                            }
                            else
                            {
                                NOTMATCH;
                            }
                            break;
                        case '+':
                            if (text[1] != reg[3])
                            {
                                if (reg[3] == text[tlen - 1])
                                {
                                    MATCH;
                                }
                                else
                                {
                                    NOTMATCH;
                                }
                            }
                            break;
                        case '?':
                            if (text[1] == reg[3] || text[2] == reg[3])
                            {
                                if (text[1] == reg[3] || text[2] == reg[3])
                                {
                                    MATCH;
                                }
                                else
                                {
                                    NOTMATCH;
                                }
                            }
                            else
                            {
                                NOTMATCH;
                            }
                             break;
                            }
                            break;
                        case '*':
                            if (reg[rlen-1] == text[tlen-1])
                            {
                                for (i = 0;i <= tlen-2;i++)
                                {
                                    if(text[i] == reg[0])
                                    {
                                        f = 1;
                                    }
                                    else
                                    {
                                        f = 0;
```

```
                  }
               }
               if ( f == 1)
               {
                  MATCH;
               }
               else
               {
                  NOTMATCH;
               }
            }
            else
            {
               NOTMATCH;
            }
            break;
      case '+' :
         if (tlen <= 2)
         {
            NOTMATCH;
         }
         else if (reg[rlen-1] == text[tlen-1])
         {
            for (i = 0;i < tlen-2;i++)
            {
               if (text[i] == reg[0])
               {
                  f = 1;
               }
               else
               {
                  f = 0;
               }
            }

            if (f == 1)
            {
               MATCH;
            }
            else
            {
               NOTMATCH;
            }
         }
            break;
      case '?':
         if (reg[rlen -1] == text[tlen-1])
         {
            MATCH;
         }
         else
         {
            NOTMATCH;
         }
      break;
   }

}
else
```

```
            printf("Does not match");
        }
        /*
         *If Regular Expression starts with '^'
         */
        else if (reg[0] == '^')
        {
            if (reg[1] == text[0])
            {
                MATCH;
            }
            else
            {
                NOTMATCH;
            }
        }
        /*
         *If Regular Expression Ends with '$'
         */
        else if (reg[rlen-1] == '$')
        {
            if (reg[rlen-2] == text[rlen-1])
            {
                MATCH;
            }
            else
            {
                NOTMATCH;
            }
        }

        else
            printf("Not Implemented");
        printf("\nDo you want to continue?(Y/N)");
        scanf(" %c", &ans);
    } while (ans == 'Y' || ans == 'y');
}
```

4.6 INPUT & OUTPUT:

<u>INPUT :</u>

```
 C.*g
Cprogramming
```

<u>OUTPUT:</u>

```
$gcc -o regex regular.c
$ ./regex

Enter the Regular Expression
C.*g

Enter the text
Cprogramming

The Text Matches The Regular Expression
Do you want to continue?(Y/N)y

Enter the Regular Expression
```

```
C*g

Enter the text
Cprogramming

The Text Doesn't match the Regular Expression
Do you want to continue?(Y/N)y

Enter the Regular Expression
C?.*g

Enter the text
Cprogramming

The Text Matches The Regular Expression
Do you want to continue?(Y/N)N
```

EXPERIMENT-5

5.1 OBJECTIVE:

Design a PDA for any given CNF. Simulate the processing of a string using the PDA and should produce a parse tree.

5.2 RESOURCE: Turbo C++

5.3 PROGRAM LOGIC :

1.Enter the CNF

2. Q is the finite number of states

∑ is input alphabet

- S is stack symbols
- δ is the transition function − Q × (∑ ∪ {ε}) × S × Q × S^*
- q_0 is the initial state (q_0 ∈ Q)
- I is the initial stack top symbol (I ∈ S)
- F is a set of accepting states (F ∈ Q)

3.Match the above from the given CNF

5.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

5.5 PROGRAM:

```
#include<iostream.h>
#include<dos.h>
#include<stdio.h>
#include<stdio.h>
#include<conio.h>

struct pdastate
        {
        int type;
        char sym;
        int trno;
        int trans[20];
        char tsym[20];
        };

class  pda
        {
        public:
                int n;
                pdastate s[30];

        pda(void)
                {
                n=0;
                }
        void show(void);
        };

void pda::show(void)
        {
        pdastate p;
        clrscr();
        for(int i=0;i<n;i++)
```

```
                {
                p=s[i];
                cout<<"
        " State No " := "<<i;
                cout<<"
        " Tag "";
                if(p.type==0)
                        cout<<"
        " Start State "";
                else
                        if(p.type==1)
                                cout<<"
        " Push "<<p.sym<<""";
                else
                        if(p.type==2)
                                cout<<"
        " Pop State "";
                else
                        if(p.type==3)
                                cout<<"
        " Read State "";
                else
                        if(p.type==4)
                                cout<<"
        " Stop State "";
                for(int j=0;j<p.trno;j++)
                        {
                cout<<"
        " Transition To State "<<p.trans[j]<<""";
                        if(p.tsym[j]!=0)
                cout<<"
        " On Symbol "<<p.tsym[j]<<"""<<endl;
                        else
                                cout<<endl;
                        }
                getch();
                if(i==5)
                clrscr();
                }
        }
union pr
        {
        char a;
        char b[2];
        };

struct prod
        {
        int type;
        union pr p;
        };
```

```
class cnf
        {
        private:
                char v[10];
                int vno;
                char t[10];
                int tno;
                struct prod p[30];
                int n;
                int vpr[10];
        public:
                cnf(void);
                pda mkpda(void);
        };

cnf:: cnf(void)
        {
        clrscr();
        char *mess[]={"-","=","["," ","C","F","A"," ","T","O"," ",
        "P","D","A"," ","]","=","-",};
        int xx=31,xxx=48,i,j;
        _setcursortype(_NOCURSOR);
        for(i=0,j=17;i<10,j>=8;i++,j--)
                {
                gotoxy(xx,1);
                cout<<mess[i];
                xx++;
                gotoxy(xxx,1);
                cout<<mess[j];
                xxx--;
                delay(50);
                }
        xx=30;xxx=49;
        _setcursortype(_NORMALCURSOR);
        cout<<"

        " Enter No Of Non Terminal Symbols ":= ";
        cin>>vno;
        for(i=0;i<vno;i++)
                {
                cout<<"
        " Enter A Non-Terminal Symbol ":=";
                cin>>v[i];
                }
        cout<<"
        " Enter The No Of Terminal Symbols ":=";
        cin>>tno;
        for(i=0;i<tno;i++)
                {
                cout<<"
```

```
            " Enter A Terminal Symbol ":=";
                        cin>>t[i];
                        }
            cout<<"
            " Enter The No Of Productions ":=";
            cin>>n;
            int count=0;
            for(i=0;i<vno;i++)
                        {
cout<<"
            " Enter No Of Productions Corrosponding To The Non-Terminal "
     <<v[i]<<" ":= ";
                        cin>>vpr[i];
                        for(int j=0;j<vpr[i];j++)
                                    {
cout<<"
            " Enter The Type Of Production <1> A-->b , <2> a-->BC ":=";
                                    cin>>p[count].type;
                                    if(p[count].type==1)
                                                {
                                                cout<<"
             "<<v[i]<<" --> ";
                                                cin>>p[count].p.a;
                                                }
                                    else
                                                {
                                                cout<<"
             "<<v[i] <<" --> ";
                                                cin>>p[count].p.b[0];
                                                cin>>p[count].p.b[1];
                                                }
                                    count++;
                                    }
                        }
            }

pda cnf:: mkpda(void)
            {
            pda p1;
            p1.s[p1.n].type=0;
            p1.s[p1.n].trno=1;
            p1.s[p1.n].trans[0]=1;
            p1.s[p1.n].tsym[0]=0;
            p1.n++;
            p1.s[p1.n].type=1;
            p1.s[p1.n].sym=v[0];
            p1.s[p1.n].trno=1;
            p1.s[p1.n].trans[0]=2;
            p1.s[p1.n].tsym[0]=0;
            p1.n++;
            p1.s[p1.n].type=2;
```

```
	p1.s[p1.n].trno=1;
	p1.s[p1.n].trans[0]=3;
	p1.s[p1.n].tsym[0]=238;
	p1.n++;
	p1.s[p1.n].type=3;
	p1.s[p1.n].trno=1;
	p1.s[p1.n].trans[0]=4;
	p1.s[p1.n].tsym[0]=238;
	p1.n++;
	p1.s[p1.n].type=4;
	p1.s[p1.n].trno=0;
	p1.n++;
	int cnt=p1.s[2].trno;
	int c1=0;
	prod temp;
	for(int i=0;i<vno;i++)
		{
		for(int j=0;j<vpr[i];j++)
			{
			temp=p[c1++];
			if(temp.type==1)
				{
				p1.s[2].trans[cnt]=p1.n;
				p1.s[2].tsym[cnt]=v[i];
				p1.s[p1.n].type=3;
				p1.s[p1.n].trno=1;
				p1.s[p1.n].trans[0]=2;
				p1.s[p1.n].tsym[0]=temp.p.a;
				p1.n++;
				cnt++;
				}
			else
				{
				p1.s[2].trans[cnt]=p1.n;
				p1.s[2].tsym[cnt]=v[i];
				p1.s[p1.n].type=1;
				p1.s[p1.n].sym=temp.p.b[1];
				p1.s[p1.n].trno=1;
				p1.s[p1.n].trans[0]=(p1.n)+1;
				p1.s[p1.n].tsym[0]=0;
				p1.n++;
				cnt++;
				p1.s[p1.n].type=1;
				p1.s[p1.n].sym=temp.p.b[0];
				p1.s[p1.n].trno=1;
				p1.s[p1.n].trans[0]=2;
				p1.s[p1.n].tsym[0]=0;
				p1.n++;
				}
			}
		p1.s[2].trno=cnt;
```

```
                    }
          return(p1);
          }

void main()
          {
          cnf c;
          pda p1;
          p1=c.mkpda();
          getch();
          p1.show();
          getch();
          }
```

5.6 INPUT & OUTPUT:

INPUT :

OUTPUT:

EXPERIMENT-6

6.1 OBJECTIVE:
Design a Lexical analyzer for identifying different types of tokens used in c language.

6.2 RESOURCE: Turbo C

6.3 PROGRAM LOGIC:
1.Enter the statement
2.Declare the identifiers and operators statically
3.Perform lexical analysis
4.Match the corresponding identifiers and operators.

6.4 PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

6.5 PROGRAM:
```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp("int",str
)==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||strcmp("static",str)==0||strcmp("switch",st
r)==0||strcmp("case",str)==0)
                printf("\n%s is a keyword",str);
        else
                printf("\n%s is an identifier",str);
}
main()
{
        FILE *f1,*f2,*f3;
        char c,str[10],st1[10];
        int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
        printf("\nEnter the c program");/*gets(st1);*/
        f1=fopen("input","w");
        while((c=getchar())!=EOF)
        putc(c,f1);
        fclose(f1);
        f1=fopen("input","r");
        f2=fopen("identifier","w");
        f3=fopen("specialchar","w");
        while((c=getc(f1))!=EOF)
        {
                if(isdigit(c))
                {
                        tokenvalue=c-'0';
                        c=getc(f1);
                        while(isdigit(c))
                        {
                                tokenvalue*=10+c-'0';
                                c=getc(f1);
                        }
                        num[i++]=tokenvalue;
                        ungetc(c,f1);
                }
                else
                if(isalpha(c))
                {
                        putc(c,f2);
                        c=getc(f1);
                        while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
                        {
                                putc(c,f2);
                                c=getc(f1);
                        }
                        putc(' ',f2);
                        ungetc(c,f1);
                }
                else
                if(c==' '||c=='\t')
                        printf(" ");
                else
                if(c=='\n')
                        lineno++;
                else
                        putc(c,f3);
        }
        fclose(f2);
        fclose(f3);
        fclose(f1);
        printf("\nThe no's in the program are");
        for(j=0;j<i;j++)
        printf("%d",num[j]);
        printf("\n");
        f2=fopen("identifier","r");
        k=0;
        printf("The keywords and identifiersare:");
        while((c=getc(f2))!=EOF)
        {
                if(c!=' ')
                        str[k++]=c;
                else
                {
                        str[k]='\0';
                        keyword(str);
                        k=0;
                }
        }
        fclose(f2);
        f3=fopen("specialchar","r");
        printf("\nSpecial characters are");
        while((c=getc(f3))!=EOF)
                printf("%c",c);
        printf("\n");
        fclose(f3);
        printf("Total no. of lines are:%d",lineno);
}
```

6.6 PRE LAB QUESTIONS

1. What is token?
2. What is lexeme?
3. What is the difference between token and lexeme?
4. Define phase and pass?
5. What is the difference between phase and pass?
6. What is the difference between compiler and interpreter?

6.7 LAB ASSIGNMENT

1. Write a program to recognize identifiers.

2. Write a program to recognize constants.
3. Write a program to recognize keywords and identifiers.
4. Write a program to ignore the comments in the given input source program.

6.8 POST LAB QUESTIONS

1. What is lexical analyzer?
2. Which compiler is used for lexical analyzer?
3. What is the output of Lexical analyzer?
4. What is LEX source Program?

6.9 INPUT & OUTPUT:

INPUT :

Enter the C program

```
X.C

/************  X.C PROGRAMM ******************/

a+b*c
```

OUTPUT:

```
The no's in the program are:

The keywords and identifiers are:

a is an identifier and terminal

b is an identifier and terminal

c is an identifier and terminal

Special characters are:

+ *

Total no. of lines are: 1
```

EXPERIMENT-7

7.1 OBJECTIVE:
Simulate a simple desktop calculator using any lexical analyzer generator tool(LEX or FLEX)

7.2 RESOURCE: Flex tool

7.3 PROGRAM LOGIC:
1.Declare the range of numbers
2.Declare the range of letters,symbols and operators
3.perform lexical analysis

7.4 PROCEDURE:
1.Goto command promt
2.Type flex filename.l
3.type gcc -lfl

7.5 PROGRAM:
LEX PROGRAM

```
%{

%}
NUMBER (([0-9]+)|([0-9]+\.[0-9]+)|([0-9]+\.[0-9]+[e][0-9]+))
%%
{NUMBER} {yylval = atof(yytext); return NUMBER;}
"+" {return *yytext;}
"-" {return *yytext;}
"*" {return *yytext;}
"/" {return *yytext;}
"\n" {return *yytext;}
. {printf("\nother default\n");}
%%
int yywrap()
{
return 1;
}
/*
int main()
{i
yylex();
}
*/
```

7.6 PRE LAB QUESTIONS:
1. List the different sections available in LEX compiler?
2. What is an auxiliary definition?
3. How can we define the translation rules?
4. What is regular expression?
5. What is finite automaton?

7.7 LAB ASSIGNMENT:
1. Write a program that defines auxiliary definitions and translation rules of Pascal tokens?
2. Write a program that defines auxiliary definitions and translation rules of C tokens?
3. Write a program that defines auxiliary definitions and translation rules of JAVA tokens

7.8 POST LAB QUESTIONS:

1. What is Jlex?
2. What is Flex?
3. What is lexical analyzer generator?
4. What is the input for LEX Compiler?
5. What is the output of LEX compiler?

7.9 INPUT & OUTPUT:

<u>INPUT :</u>

```
 2+3
```

<u>OUTPUT:</u>

```
    5
Another default
```

EXPERIMENT-8

8.1 OBJECTIVE:

Program to recognize the identifiers, if and switch statements of C using a lexical analyzer generator tool.

8.2 RESOURCE: Flex tool

8.3 PROGRAM LOGIC:

1.Declare the range of identifiers, if and switch statements of C
2.Use flex software
3..Perform lexical analysis

8.4 PROCEDURE:

1.Goto command promt
2.Type flex   filename.l
3.type gcc -lfl

8.5 PROGRAM

```
%{
%}
ID [a-zA-Z][a-zA-Z0-9]*
NUMBER ([0-9]+)|([0-9]+\.[0-9]+)
OP "+"|"-"|"*"|"/"|"--"|"&&"|"||"|">"|"<"|"=="|">="|"<="|"="
%%
SWITCH {/*printf("It is switch\n");*/ return SWITCH; }
CASE {/*printf("It is case\n");*/ return CASE; }
BREAK {/*printf("It is case\n");*/ return BREAK; }
DEFAULT {/*printf("It is case\n");*/ return DEFAULT; }
{ID} {/*printf("It is id\n");*/ return ID; }
{NUMBER} {/*printf("It is number\n");*/ return NUMBER; }
{OP} {/*printf("It is operator\n");*/ return OP; }
"(" {return *yytext;}
")" {return *yytext;}
"{" {return *yytext;}
"}" {return *yytext;}
";" {return *yytext;}
":" {return *yytext;}
"\n" {return *yytext;}
[ \t] {}
. {return *yytext;}
%%
/*
int main()
{
yyin = fopen("test","r");
yylex();
}
*/
int yywrap()
{
return 1;
}
```

8.6 INPUT & OUTPUT:

<u>INPUT :</u>

```
 #include<stdio.h>
Main()
{
Int a=1;
}
```

OUTPUT:

```
Keywords:include,main()
Constantd:1
Datatypes:int
Deliminators:{ }
Relational operators:< >
```

EXPERIMENT-9

9.1 OBJECTIVE:

Consider the following grammar:

S→ABC
A→abA|ab
B→b|BC
C→c|cC

Design any shift reduced parser which accepts a string and tells whether the string is accepted by above grammar or not.

9.2 RESOURCE: Turbo c

9.3 PROGRAM LOGIC:

1.Enter the string
2.Compare the string with the grammar
3.Shift reduce the grammar
4.If accept then string is accepted
5.Else string is not accepted

9.4 PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

9.5 PROGRAM:

```
#include"stdio.h"
#include"stdlib.h"
#include"conio.h"
#include"string.h"
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();
void main()
{
clrscr();
printf("\n\t\t SHIFT REDUCE PARSER\n");
printf("\n GRAMMER\n");
printf("\n E->E+E\n E->E/E");
printf("\n E->E*E\n E->a/b");
printf("\n enter the input symbol:\t");
gets(ip_sym);
printf("\n\t stack implementation table");
printf("\n stack\t\t input symbol\t\t action");
printf("\n______\t\t ____________\t\t ______\n");
printf("\n $\t\t%s$\t\t\t--",ip_sym);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
len=strlen(ip_sym);
for(i=0;i<=len-1;i++)


{
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]=' ';
ip_ptr++;
printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
check();
st_ptr++;
}
st_ptr++;
check();
}
void check()
{
int flag=0;
temp2[0]=stack[st_ptr];
temp2[1]='\0';
if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
{
 stack[st_ptr]='E';
 if(!strcmpi(temp2,"a"))
  printf("\n $%s\t\t%s$\t\t\tE->a",stack, ip_sym);
 else
  printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip_sym);
 flag=1;
}
if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))
{
 flag=1;
}
if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E*E")))
{
strcpy(stack,"E");
st_ptr=0;
if(!strcmpi(stack,"E+E"))
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
else
if(!strcmpi(stack,"E\E"))
printf("\n $%s\t\t %s$\t\t\tE->E\E",stack,ip_sym);
else
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
flag=1;
}

if(!strcmpi(stack,"E")&&ip_ptr==len)
{
printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n%s\t\t\t%s\t\t reject",stack,ip_sym);
exit(0);
}
return;
}
```

9.6 PRE-LAB QUESTIONS

1 Why bottom-up parsing is also called as shift reduce parsing?

2 What are the different types of bottom up parsers?
3 What is mean by LR (0) items?
4 Write the general form of LR(1) item?
5 What is YACC?

9.7 LAB ASSIGNMENT

1 Write a program to compute FOLLOW for the following grammar?
E□□TE'
E'□+TE'/î
T□FT’
T'□*FT'/î
F□(E)/i
2 Write a program to construct LALR parsing table for the following grammar.
S□iCtSS’
S’□eS/ î

9.8 POST-LAB QUESTIONS:

1. What is LALR parsing?
2. What is Shift reduced parser?
3. What are the operations of Parser?
4. What is the use of parsing table?
5. What is bottom up parsing

9.9 INPUT & OUTPUT:

INPUT :

```
 SHIFT REDUCE PARSER
 GRAMMER
E->E-E
E->E/E
E->E*E
E->E/e
E-> a/b
Enter the input symbol    a+b
```

OUTPUT:

```
                         Stack  implementation table

Stack               input symbol           action
______              __________             _______
$                   a+b$                     ----
$a                  +b$                    shift a
$E                  +b$                    E->a
$E+                   b$                   shift +
$E+b                   $                   shift b
$E+E                   $                   E->b
$E                     $                   E->E+E
$E+                    $                   ACCEPT
```

EXPERIMENT-10

10.1 OBJECTIVE:

Design a YACC program that reads a C program from input file and identify all valid C identifiers and for loop statements.

10.2 RESOURCE:YACC Tool

10.3 PROGRAM LOGIC:

1.Declare the range of c identifiers and loop statements
2.Perform lexical analysis
3.Enter the c program

10.4 PROCEDURE:

1.Goto command promt
2.Type flex filename.l
3.type gcc -lfl

10.5 PROGRAM:

// **Lex file: for.l**

```
alpha [A-Za-z]
digit [0-9]

%%

[\t \n]
for             return FOR;
{digit}+    return NUM;
{alpha}({alpha}|{digit})* return ID;
"<="        return LE;
">="        return GE;
"=="        return EQ;
"!="        return NE;
"||"        return OR;
"&&"         return AND;
.           return yytext[0];

%%
```

// **Yacc file: for.y**

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM FOR LE GE EQ NE OR AND
%right "="
%left OR AND
%left '>' '<' LE GE EQ NE
%left '+' '-'
%left '*' '/'
%right UMINUS
%left '!'

%%

S        : ST {printf("Input accepted\n"); exit(0);}
ST       : FOR '(' E ';' E2 ';' E ')' DEF
         ;
```

```
DEF    : '{' BODY '}'
          | E';'
          | ST
          |
          ;
BODY  : BODY BODY
          | E ';'
          | ST
          |
          ;
       E        : ID '=' E
         | E '+' E
         | E '-' E
         | E '*' E
         | E '/' E
         | E '<' E
         | E '>' E
         | E LE E
         | E GE E
         | E EQ E
         | E NE E
         | E OR E
         | E AND E
         | E '+' '+'
         | E '-' '-'
         | ID
         | NUM
         ;


E2     : E'<'E
        | E'>'E
        | E LE E
        | E GE E
        | E EQ E
        | E NE E
        | E OR E
        | E AND E
        ;
%%

#include "lex.yy.c"
main() {
   printf("Enter the expression:\n");
   yyparse();}
```

10.6 INPUT & OUTPUT:

INPUT :

```
nn@linuxmint ~ $ lex for.l
nn@linuxmint ~ $ yacc for.y
conflicts: 25 shift/reduce, 4 reduce/reduce
nn@linuxmint ~ $ gcc y.tab.c -ll -ly
nn@linuxmint ~ $ ./a.out
Enter the expression:
for(i=0;i<n;i++)
i=i+1;
```

OUTPUT:

```
Input accepted
```

EXPERIMENT-11(a)

11.1 OBJECTIVE:
Program to eliminate left recursion and left factoring from the given CFG**.**
11.2 RESOURCE : Turbo C

11.3 PROGRAM LOGIC:

Assign an ordering *A*1,…,*An* to the nonterminals of the grammar.

```
for $i$:=1 to $n$ do begin
for $j$:=1 to $i$−1 do begin
for each production of the form $A_i \to A_j\alpha$ do begin
remove $A_i \to A_j\alpha$ from the grammar
for each production of the form $A_j \to \beta$ do begin
add $A_i \to \beta\alpha$ to the grammar
end
end
end
transform the $A_i$-productions to eliminate direct left recursion
end
```

11.4 PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

11.5 PROGRAM
```
#include<stdio.h>
#include<string.h>
  #define SIZE 10
  int main () {
       char non_terminal;
      char beta,alpha;
       int num;
      char production[10][SIZE];
     int index=3; /* starting of the string following "->" */
       printf("Enter Number of Production : ");
       scanf("%d",&num);
       printf("Enter the grammar as E->E-A :\n");
       for(int i=0;i<num;i++){
            scanf("%s",production[i]);
       }
       for(int i=0;i<num;i++){
            printf("\nGRAMMAR : : : %s",production[i]);
            non_terminal=production[i][0];
            if(non_terminal==production[i][index]) {
                 alpha=production[i][index+1];
                 printf(" is left recursive.\n");
                 while(production[i][index]!=0 && production[i][index]!='|')
                      index++;
                 if(production[i][index]!=0) {
                     beta=production[i][index+1];
                      printf("Grammar without left recursion:\n");
                     printf("%c->%c%c\'",non_terminal,beta,non_terminal);
                      printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
```

```
                }
                else
                    printf(" can't be reduced\n");
            }
            else
                printf(" is not left recursive.\n");
            index=3;
        }
    }
```

11.6 INPUT & OUTPUT:

INPUT :

```
 Enter no. of Productions:   4
 Enter the Grammer as E->E-A :
 E->EA | A
A-> AT | a
T->a
E->i
```

OUTPUT:

```
GRAMMER: : :  E->EA | A  is left  recursive
Grammar without left recursion
E->AE’
E-‘>AE’|E

GRAMMER: : :  A-> AT | a  is left  recursive
Grammar without left recursion
A->aA’
A’->TA’ | E

GRAMMER: : :  T->a   is non left  recursive
GRAMMER: : :  E->i   is left  recursive
```

EXPERIMENT-11(b)

11.1 OBJECTIVE:
Program for left factoring
11.2 RESOURCE : Turbo C
11.3 PROGRAM LOGIC:

For each non terminal A find the longest prefix □ common to two or more of its alternatives. If □!= E, i.e., there is a non trivial common prefix, replace all the A productions
A==>□□1|□□2|..............|□□n|□□□□where □ represents all alternatives that do not begin with □ by
A==>□A'|□
A'==>□1□|2|.............|□n
Here A' is new non-terminal. Repeatedly apply this transformation until no two alternatives for a non-terminal have a common prefix.

11.4 PROCEDURE:
Go to debug -> run or press CTRL + F9 to run the program

11.5 PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char a[10],a1[10],a2[10],a3[10],a4[10],a5[10];
int i,j=0,k,l;
clrscr();
printf("enter any productions A->");
gets(a);
for(i=0;a[i]!='/';i++,j++)
a1[j]=a[i];
a1[j]='\0';
for(j=++i,i=0;a[j]!='\0';j++,i++)
a2[i]=a[j];
a2[i]='\0';
k=0;
l=0;
for(i=0;i<strlen(a1)||i<strlen(a2);i++)
{
if(a1[i]==a2[i])
{
a3[k]=a1[i];
k++;
}
else
{
a4[l]=a1[i];
a5[l]=a2[i];
l++;
}}
a3[k]='X';
a3[++k]='\0';
a4[l]='/';
a5[l]='\0';
a4[++l]='\0';
strcat(a4,a5);
printf("\n A->%s",a3);
printf("\n X->%s",a4);
getch();
}
```

11.10 INPUT & OUTPUT:

INPUT :

```
 Enter any Productions

 A->bcd/bcf
```

OUTPUT:

```
A->dcX
X->d/f
```

EXPERIMENT-12

12.1 OBJECTIVE:

YACC program that reads the input expression and convert it to postfix expression.

12.2 RESOURSE:

12.3 PROGRAM LOGIC:

```
1. Create a stack
2. for each character ‘t’ in the input stream {
      o if (t is an operand)
        output t
      o else if (t is a right parentheses){
        POP and output tokens until a left parentheses is popped(but do not output)
        }
      o else {
        POP and output tokens until one of lower priority than t is encountered or a left parentheses is encountered
        or the stack is empty
        PUSH t
        }
3. POP and output tokens until the stack is empty
```

12.4 PROCEDURE:

1.Goto command promt
2.Type flex   filename.l
3.type gcc -lfl

12.5 PROGRAM:

```
********************* inpost.l **********************
%{

%{
#include<stdio.h>
#include<math.h>
#include "y.tab.h"
%}
%%
[0-9]+ {
 yylval.dval=yytext[0];
 return NUMBER;
 }
[t];
n return 0;
. {return yytext[0];}
%%
void yyerror(char *str)
{
 printf("n Invalid Character...");
}
int main()
{
 printf("Enter Expression => ");
 yyparse();
 return(0);
}
********************* inpost.y *********************
%{
#include<stdio.h>
int yylex(void);
int k=0;
int i;
char sym[26];
FILE *fp;
%}
%union
{
 char dval;
}
%token <dval> NUMBER
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS
%type <dval> state
%type <dval> exp
%%
state : exp {
 printf("nConverted Postfix expression is => ");
 fp=fopen("postfix.txt","w
 for(i=0;i<k;i++)
 {
 fprintf(fp,"%c",sym[i]);
 printf("%c",sym[i]);
 }
 fclose(fp);
 }
 ;
```

```
exp : NUMBER {$$=$1;sym[k]=(char)$$;k++}
 | exp '+' exp {sym[k]='+';k++}
 | exp '-' exp {sym[k]='-';k++}
 | exp '*' exp {sym[k]='*';k++}
 | exp '/' exp {sym[k]='/';k++}
 ;
%%
```

12.9 INPUT & OUTPUT:

<u>INPUT :</u>

```
[a40@localhost ~]$ lex inpost.l
[a40@localhost ~]$ yacc -d inpost.y
[a40@localhost ~]$ cc lex.yy.c y.tab.c -ll
[a40@localhost ~]$ ./a.out

Enter Expression => 7*9+6/2-1
```

<u>OUTPUT :</u>

```
Converted Postfix expression is => 79*62/+1-
```

EXPERIMENT-13

13.1 OBJECTIVE:

YACC program that finds C variable declarations in C source file and save them into the symbol table, which is organized using binary search tree**.**

13.2 RESOURSE:YACC Tool

13.3 PROGRAM LOGIC:

1.Enter the c program
2.Check whether c variable declarations are present.
3.Check the symbol table positions
4.Allow the symbol entries.

13.4 PROCEDURE:

1.Goto command promt
2.Type flex filename.l
3.type gcc –lfl

13.5 PROGRAM:

```
%{
#include
#include "y.tab.h"
#include
int fl=0,i=0,type[100],j=0,error_flag=0;
char symbol[100][100],temp[100];
%}
%token INT FLOAT C DOUBLE CHAR ID NL SE O
%%
START:S1 NL {return;}
;
S1:S NL S1
|S NL
;
S: INT  L1 E
|FLOAT L2 E
|DOUBLE L3 E
|CHAR L4 E
|INT L1 E S
|FLOAT L2 E S
|DOUBLE L3 E S
|CHAR L4 E S
|O
;
L1:L1 C ID {strcpy(temp,(char *)$3);insert(0);}
|ID {strcpy(temp,(char *)$1);insert(0);}
;
L2:L2 C ID {strcpy(temp,(char *)$3);insert(1);}
|ID {strcpy(temp,(char *)$1);insert(1);}
;
L3:L3 C ID {strcpy(temp,(char *)$3);insert(2);}
|ID {strcpy(temp,(char *)$1);insert(2);}
;
L4:L4 C ID {strcpy(temp,(char *)$3);insert(3);}
|ID {strcpy(temp,(char *)$1);insert(3);}
;
E:SE
;
```

```
%%
main()
{
yyparse();
if(error_flag==0)
for(j=0;j
{
if(type[j]==0)
printf(" INT - ");
if(type[j]==1)
printf(" FLOAT - ");
if(type[j]==2)
printf(" DOUBLE - ");
if(type[j]==3)
printf(" CHAR - ");
printf(" %s\n",symbol[j]);
}
}
void yyerror()
{ printf("SYNTAX ERROR\n");
error_flag=1;
}
void insert(int type1)
{
fl=0;
for(j=0;j
if(strcmp(temp,symbol[j])==0)
{
if(type[i]==type1)
printf("REDECLARATION OF %s\n",temp);
else
{
printf("MULTIPLE DECLARATION OF %s\n",temp);
error_flag=1;
}
fl=1;
}
if(fl==0)
{
strcpy(symbol[i],temp);
type[i]=type1;
i++;
}
}
```

13.6 INPUT & OUTPUT:

```
char tt
SYNTAX ERROR
float gg;
char dd,ff;
int ll;
FLOAT - gg
CHAR - dd
CHAR - ff
INT - ll
int xx;
float xx;

MULTIPLE DECLARATION OF xx
```

EXPERIMENT-14

14.1 OBJECTIVE:

YACC program that reads the C statements from an input file and converts them into quadruple three address intermediate code**.**

14.2 RESOURSE:YACC Tool

14.3 PROGRAM LOGIC:

1.Enter the instruction
2.Divide the instruction into t1,t2,t3…..
3.Check the operand1,operand2,operator and result
4.If operand is equal to minus,don’t consider the operand

14.4 PROCEDURE:

1.Goto command promt
2.Type flex filename.l
3.type gcc –lfl

14.5 PROGRAM:

CODE :

```
/* LEX FILE */

%{
#include “y.tab.h”
extern char yyval;
%}

NUMBER [0-9]+
LETTER [a-zA-Z]+

%%


{NUMBER} {yylval.sym=(char)yytext[0]; return NUMBER;}
{LETTER} {yylval.sym=(char)yytext[0];return LETTER;}
\n {return 0;}
. {return yytext[0];}

%%

/* yacc file */

%{

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void ThreeAddressCode();
void triple();
void qudraple();
char AddToTable(char ,char, char);

int ind=0;
char temp=’A';
struct incod
{
char opd1;
char opd2;
char opr;
};
%}

%union
{
char sym;
}

%token <sym> LETTER NUMBER
%type <sym> expr
%left ‘-”+’
%right ‘*”/’

%%

statement: LETTER ‘=’ expr ‘;’ {AddToTable((char)$1,(char)$3,’=');}
| expr ‘;’
;

expr: expr ‘+’ expr {$$ = AddToTable((char)$1,(char)$3,’+');}
| expr ‘-’ expr {$$ = AddToTable((char)$1,(char)$3,’-');}
| expr ‘*’ expr {$$ = AddToTable((char)$1,(char)$3,’*');}
| expr ‘/’ expr {$$ = AddToTable((char)$1,(char)$3,’/');}
| ‘(‘ expr ‘)’ {$$ = (char)$2;}
| NUMBER {$$ = (char)$1;}
| LETTER {$$ = (char)$1;}
;

%%

yyerror(char *s)
{
printf(“%s”,s);
exit(0);
}
```

```
struct incod code[20];

int id=0;

char AddToTable(char opd1,char opd2,char opr)
{
code[ind].opd1=opd1;
code[ind].opd2=opd2;
code[ind].opr=opr;
ind++;
temp++;
return temp;
}

void ThreeAddressCode()
{
int cnt=0;
temp++;
printf("\n\n\t THREE ADDRESS CODE\n\n");
while(cnt<ind)
{
printf("%c : = \t",temp);
if(isalpha(code[cnt].opd1))
printf("%c\t",code[cnt].opd1);
else
{printf("%c\t",temp);}

printf("%c\t",code[cnt].opr);

if(isalpha(code[cnt].opd2))
printf("%c\t",code[cnt].opd2);
else
{printf("%c\t",temp);}

printf("\n");
cnt++;
temp++;
}
}

void quadraple()
{
int cnt=0;
temp++;
printf("\n\n\t QUADRAPLE CODE\n\n");
```

```
while(cnt<ind)
{
//printf("%c : = \t",temp);
printf("%d",id);
printf("\t");
printf("%c",code[cnt].opr);
printf("\t");
if(isalpha(code[cnt].opd1))
printf("%c\t",code[cnt].opd1);
else
{printf("%c\t",temp);}

//printf("%c\t",code[cnt].opr);

if(isalpha(code[cnt].opd2))
printf("%c\t",code[cnt].opd2);
else
{printf("%c\t",temp);}

printf("%c",temp);

printf("\n");
cnt++;
temp++;
id++;

}
}

void triple()
{
int cnt=0,cnt1,id1=0;
temp++;
printf("\n\n\t TRIPLE CODE\n\n");
while(cnt<ind)
{
//printf("%c : = \t",temp);

if(id1==0)
{
printf("%d",id1);
printf("\t");
printf("%c",code[cnt].opr);
printf("\t");
if(isalpha(code[cnt].opd1))
printf("%c\t",code[cnt].opd1);
else
{printf("%c\t",temp);}

//printf("%c\t",code[cnt].opr);
cnt1=cnt-1;
if(isalpha(code[cnt].opd2))
printf("%c",code[cnt].opd2);
else
{printf("%c\t",temp);}
}
else
{
printf("%d",id1);
printf("\t");
printf("%c",code[cnt].opr);
printf("\t");
if(isalpha(code[cnt].opd1))
printf("%c\t",code[cnt].opd1);
else
{printf("%c\t",temp);}

//printf("%c\t",code[cnt].opr);
cnt1=cnt-1;
if(isalpha(code[cnt].opd2))
printf("%d",id1-1);
else
{printf("%c\t",temp);}
}

printf("\n");
cnt++;
temp++;
id1++;

}

}

main()
{
printf("\nEnter the Expression: ");
yyparse();
temp='A';
ThreeAddressCode();
```

```
quadraple();
triple();
}

yywrap()
{
return 1;
}
```

14.6 INPUT & OUTPUT:

```
administrator@ubuntu:~/Desktop$ flex th.l
administrator@ubuntu:~/Desktop$ yacc -d th.y
administrator@ubuntu:~/Desktop$ gcc lex.yy.c y.tab.c -ll -lm
administrator@ubuntu:~/Desktop$ ./a.out

Enter the Expression: a=((b+c)*(d+e))
syntax error

administrator@ubuntu:~/Desktop$ ./a.out

Enter the Expression: a=((b+c)*(d/e));
THREE ADDRESS CODE

B : = b + c
C : = d / e
D : = B * C
E : = a = D
QUADRAPLE CODE

0 + b c G
1 / d e H
2 * B C I
3 = a D J
TRIPLE CODE

0 + b c
1 / d 0
2 * B 1
3 = a 2
```