Index Of The Experiments Performed

S.no	Experiment Description	Page No.	Experiment Date	Submission Date	Remarks
1.	Implementation of multi D array	02	31/07/19		
2.	Implementation of Lexical Analyzer		07/08/19		
3	check whether a string belongs to a grammar		14/8/19		
4.	FIRST & FOLLOW of non-terminals		21/08/19		
5.	find leading terminals.		28/8/19		
6.	find trailing terminals		4/9/19		
7.	Show all operations of a stack		4/9/19		
8.	Show various operations (read, write & modify)		10/9/19		
9.	Write a program to check whether a grammar is left recursive & remove left recursion		6/11/19		
10.	WAP to remove left factoring		6/11/19		
11.	grammar is operator precedent		6/11/19		
12.	generate a parse tree		6/12/19.		

```
Enter 12 values:
1
2
3
4
5
6
7
8
9
10
1
12

Displaying values:
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 1
test[1][2][1] = 12
```

Experiment : 01

Date

Page No.

- **Aim :-** Write a program for implementation of multi-D array.
- **Software Required :-** C/C++ Compiler, Notepad
- **Theory.**

In C programming, creating array of an array is called a ~~multi~~ multidimensional array.
For example:-

```
float x[3][4];
```

Here, x is a 2D array (two-dimensional). The array can hold 12 elements, 3 rows and each row with 4 columns.

- **Program**

```
#include <stdio.h>
int main()
{ int test[2][3][2];
  printf("Enter 12 values : \n");
  for(int i=0; i<2; ++i)
  { for(int j=0; j<3; ++j)
    { for(int k=0; k<2; ++k)
      { scanf("%d", &test[i][j][k]);
      }
    }
}
```

Experiment :

Date

Page No.

```
printf("\n Displaying values : \n");
for(int i = 0; i<2; ++i)
{ for(int j=0; j<3; ++j)
  { for(int k=0; k<2; ++k)
    { printf("test[%d][%d][%d] = %d \n", i, j, k,
                test[i][j][k]);
    }
  }
}
return 0;
}
```

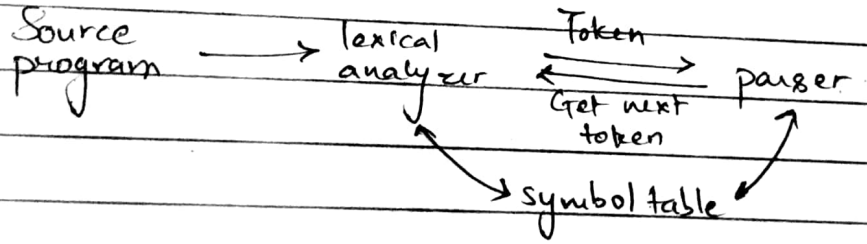
Experiment : 02

Date
Page No.

- **Aim :-** Write a program to implement ~~NFA to DFA Conversion~~. Lexical Analyzer
- **Software Required :-** C/C++ Compiler, Notepad
- **Theory :-**

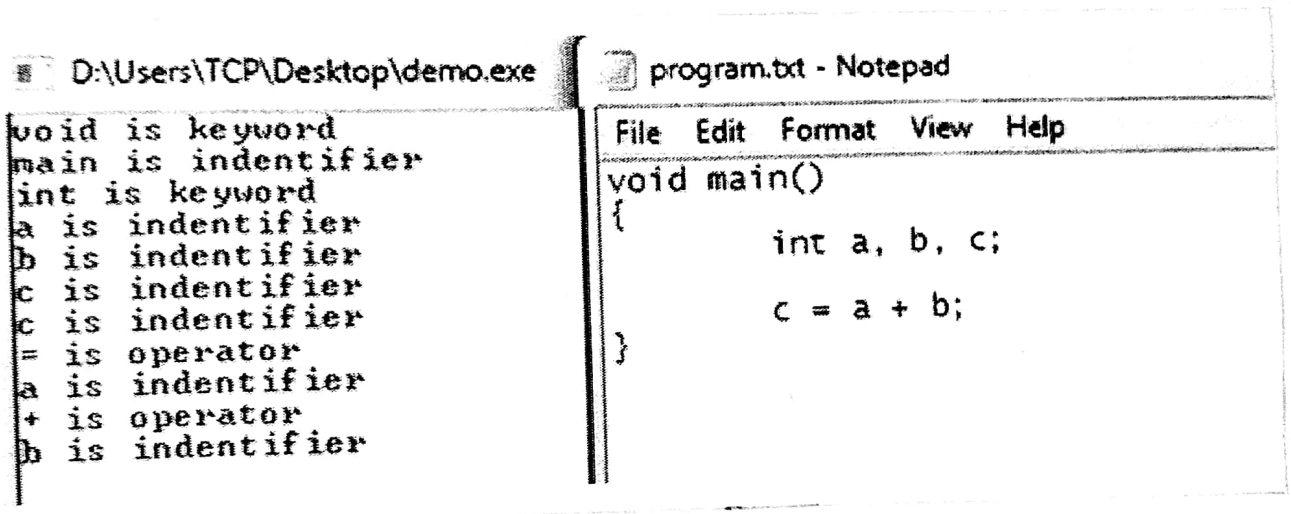
Compiler is responsible for converting high level lang. in machine language. There are several phases involved in this & lex analysis is the first phase.

Lexical analyzer reads the characters from source code and cover it into tokens.



- **Program**

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
#include <ctype.h>
int isKeyword(char buffer[])
{ char keywords[32][10] = {"auto","break","case","const","continue",
                           "default","do","double","else","enum",
                           "extern","float","for","goto","void"};
```



Experiment :

Date
Page No.

```
int i, flag = 0;
for(i=0; i<32; ++i)
{ if(strcmp(keywords[i], buffer == 0)
    { flag = 1;
      break;
    }
}
return flag;
}
int main()
{ char ch, buffer[15], operators[] = "+-*/%=";
  FILE *fp;
  int i, j=0;
  fp = fopen("program.txt", "r");
  if (fp == NULL)
  { printf("error while opening the file \n");
    exit(0);
  }
while((ch = fgetc(fp)) != EOF)
{ for(i=0; i<6; ++i)
  { if (ch == operators[i])
    printf("%c is operator\n", ch);
  }
  if (isalnum(ch))
  { buffer[j++] = ch;
  }
else if((ch == ' ' || ch == '\n') && (j != 0))
{ buffer[j] = '\0';
```

Experiment :

Date

Page No.

```
        j=0;
        if(isKeyword(buffer)==1)
            printf("%s is Keyword \n", buffer);
        else
            printf("%s is indentifier \n", buffer);
    }
}
    fclose(fp);
    return 0;
}
```

```
The grammer is:
 S->aS
 S->Sb
 S->ab
Enter the string to be checked:
aab
String accepted....!!!!
```

Expertment : 05

Date
Page No.

- **Aim :-** Write a program to check whether a string belongs to a grammar or not.
- **Software Required :-** C/C++ Compiler, Notepad
- **Theory** Program

```
#include <stdio.h>
#include <conio.h>
#include <conio.h>
void main()
{   char string[50]
    int flag, count=0;
    clrscr();
    printf("The grammar is : S→aS, S→Sb, S→ab\n");
    printf("Enter the string to be checked: \n");
    gets(string);
    if(string[0]=='a')
    {   flag=0;
        for(count=1; string[count-1]!='\0'; count++)
        {   if(string[count]=='b')
            {   flag=1;
                continue;
            }else if((flag==1)&&(string[count]=='a'))
            {   printf("String doesn't belong to specified
                        grammar");
                break;
```

Experiment :

Date

Page No.

```
} else if (string[count] == 'a')
continue;
else if (flag == 1) && (string[count] == '\0'))
{ printf ("\n String accepted ... !!! ");
  break;
}
else
{ printf (" String not accepted ");
}
}
} getch();
}
```

Experiment : 04

Date
Page No.

- Aim :- Write a program to compute FIRST of non-terminals & FOLLOW of non-terminal
- Software Required :- C/C++ Compiler
- Theory :

FIRST is applied to the RHS of a production rule, and tells us all the terminal symbols that can start sentences derived from the RHS.

① ~~For any~~ FIRST(a) = {a}
& FIRST(ϵ) = {ϵ}

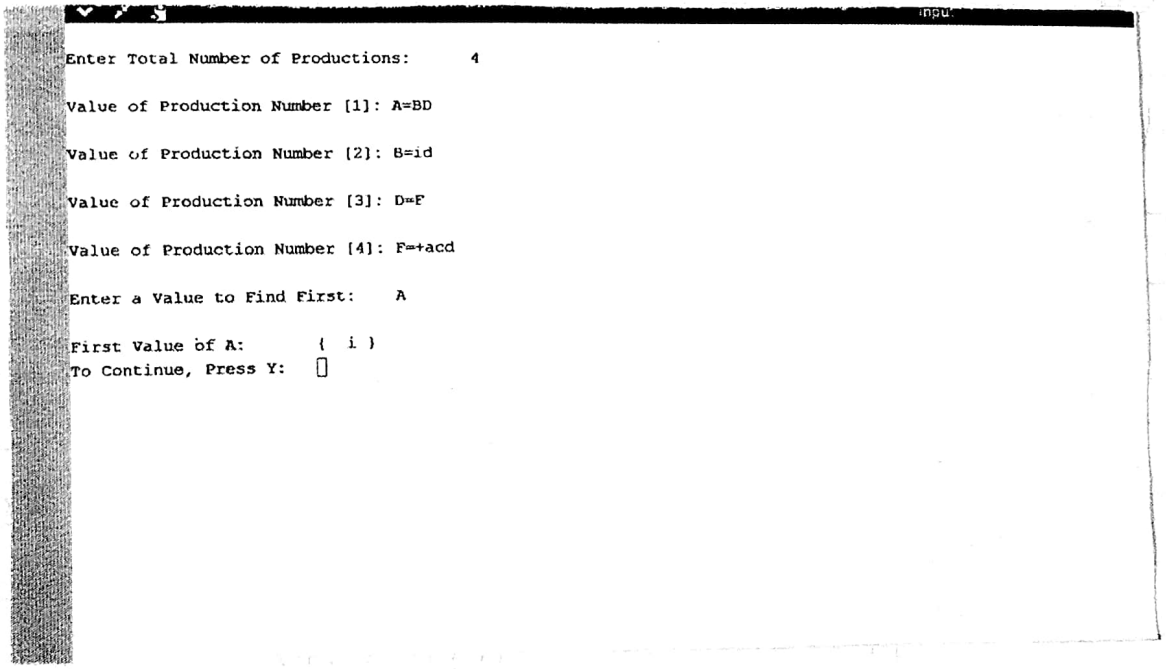
② $A \rightarrow a_1 | a_2 | \cdots | a_n$
FIRST(A) = FIRST(a_1) ∪ FIRST(a_2) ∪ ... ∪ FIRST(a_n)

③ $A \rightarrow \beta_1 \beta_2 \cdots \beta_n$ (β_i is terminal or non terminal)
FIRST(β_1) is in FIRST($\beta_1 \beta_2 \cdots \beta_n$)
if β_1 derives ϵ then FIRST(β_2) and so on
OR
ϵ is in FIRST($\beta_1 \beta_2 \cdots \beta_n$) only if ϵ is in FIRST($\beta_i$), for all $0 \leq i \leq n$.

~~Program~~

FOLLOW is used only if the current non-terminal can derive ϵ ; then we're interested in what could have followed it in a sentential form.

1. If S is the start symbol, then put ⊥ into FOLLOW(S)
2. Examine all rules of the form $A \rightarrow \alpha X \beta$; then



Experiment :

Date
Page No.

a) FIRST(β) is in FOLLOW(X)
b) If β can derived the empty string then put FOLLOW(A) into FOLLOW(X).

- Code

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
int n, m=0, p, i=0, j=0;
char a[10][10], f[10];
void follow (char c);
void first (char c);
int main ()
{   int i,z;
    char c, ch;
    printf ("Enter the no of productions: \n");
    scanf ("%d", &n);
    printf ("Enter the productions: \n");
    for(i=0; i<n; i++)
    scanf ("%s %c", a[i], &ch);
    do
    {   m=0
        printf ("Enter the elements whose first and follow
                 is to be found: ");
        scanf ("%c", &c);
        first(c);
```

Experiment :

Date

Page No.

```
printf ("FIRST (%c) = { ", c);
for (i = 0 ; i<m ; i++)
printf ("%c", f[i]);
printf ("} \n");
strcpy (f, " ");
m = 0;
follow (c);
printf ("follow (%c) = { ", c);
for (i = 0; i<m ; i++)
printf ("%c" , f[i]);
printf ("} \n");
printf ("continue (0/1) ? ");
scanf ("%d %c", &z , &ch);
}
while (z == 1);
return 0;
}
void first (char c)
{ int k;
if (! isupper (c))
f[m++] = c;
for (k=0; k<n; k++)
{ if (a[k][0] == c)
{ if (a[k][2] == '$')
follow [a[k][0]);
else if (is lower (a[k][2]))
f[m++] = a[k][2];
else first (a[k][2]);
}
```

Experiment :

Date

Page No.

```
    }
}
void follow (char c)
{ if(a[0][0] == c)
    f[m++] = '$';
    for(i=0; i<n; i++)
    { for(j=2; j<strlen(a[j]); j++)
        { if(a[i][j] == c)
            { if(a[i][j+1] != '\0')
                first(a[i][j++]);
                if(a[i][j++] == '\0' && c1 = a[i][0])
                follow(a[i][0]);
            }
        }
    }
}
```

Experiment : 05

Date

Page No.

- Aim : Write a program to find leading terminals.
- Software required :- C/C++ compiler, notepad
- Code :

```
#include <conio.h>
#include<stdio.h>
char arr[18][3] = {{'E', '+', 'T'}, {'E', '*', 'f'},
  {'E', '(', 'F'}, {'E', ')', 'f'}, {'E', 'i', f'}, {'E',
  'i', 'F'}, {'f', '+', 'F'}, {F', '*', 'f'}, {'f', '(',
  'f'}, {'f', ')', 'F'}};
char prod[6] = "EETTFF"
char res[6][3] = {{'E', '+', 'T'}, {'T', '\0'}, {'T', '*',
  'f'}, {'F', '\0'}, {'(', 'E', ')'}, {'i', '\0'}};
char stack[5][2];
int top = -1;
void install (char pro, char re)
{ int i;
  for (i=0; i<18; ++i)
  { if (arr[i][0] == pro && arr[i][1] == re)
    { arr[i][2] = 'T';
      break;
    }
  }
  ++top;
  stack[top][0] = pro;
```

E	+	T
E	*	T
E	(	T
E	)	F
E	i	T
F	$	F
F	+	F
F	*	F
F	(	T
F	)	F
T	i	T
T	$	F
T	+	F
T	*	T
T	(	T
T	)	F
T	i	T
T	$	F

E -> + * (i
F -> (i
T -> * (i

Experiment :

Date
Page No.

```
    stack [top][1] = re;
};
void main()
{ int i = 0, j;
  char pro, re, pri = ' ';
  clrscr();
  for (i=0; i<6; ++i)
  { for (j=0; j<3 && res[i][j] != '0'; ++j)
    { if (res[i][j] == '+' || res[i][i] == '*' ||
          res[i][j] == '(' || res[i][j] == ')' ||
          res[i][j] == 'i' || res[i][j] == '$')
      { install (prod[i], res[i][j]);
        break;
      }
    }
  }
while (to >= 0)
{ pro = stack [top][0];
  re = stack [top][1];
  --top;
  for (i=0; i<6; ++i)
  { if res[i][0] == pro && res[i][0] != prod[i])
    { install (prod[i], re);
    }
  }
}
for (i=0; i<18; ++i)
{ printf ("\n\t");
```

Experiment :

Date

Page No.

```
    for (j=0; j<3 ; ++j)
        printf("%c\t", arr[i][j]);
    }

    getch();
    clrscr();
    printf ("\n\n");
    for(i=0; i< 18 ; ++i)
    { if (pri != arr[i][0])
        { pri = arr[i][0];
          printf ("\n\t %c -> ", pri);
        }
        if (arr[i][2] == 'T')
            printf("%c ", arr[i][j]);
    }
    getch();
}
```

Experiment: 06

Date

Page No.

- Aim:- Write a program to find the trailing of terminals
- Software required:- C/C++ compiler, notepad
- Code:

```
#include <conio.h>
#include<stdio.h>
char arra[18][3] = {{'E','+','f'},{'E','*','f'},
    {'E','(','F'},{'E',')','F'},{'E','i','F'},
    {'E','$','F'},{'F','+','f'},{'f','*','F'},
    {'F','$','F'},{'T','+','F'},{'T','*','f'},
    {'T','(',':F'},{'T',')','F'},{'T','i','F'},
    {'T','$','f'}};
char prod[6] = "EETTff";
char res[6][3] = {{'E','+','T'},{'T','\0','\0'},
  {'T','*','F'},{'F','\0','\0'},{'(','E',')'}};
char stack[5][2];
int top = -1;
void install(char pro, char re)
{ int i;
  for(i=0; i<18; ++i)
  { if arr[i][0] == pro && arr[i][1] == re)
    { arr[i][2] = 'T';
      break;
    }
    ++top;
```

E	+	T
E	*	T
E	(	F
E	)	T
E	i	T
E	$	F
F	+	F
F	*	F
F	(	F
F	)	T
F	i	T
F	$	F
T	+	F
T	*	T
T	(	F
T	)	T
T	i	T
T	$	F

E -> + *) i
F ->) i
T -> *) i _

```
        arr[i][2] = 'T';
        break;
        stack[top][0] = pro;
        stack[top][1] = re;
}
void main()
{   int i = 0, j;
    char pro, re, pri = '';
    clrscr();
    for(i = 0; i<6; ++i)
    {   for(j = 2; j>=0; --j)
        {   if(res[i][j] == '+' || res[i][j] = '*' ||
            res[i][j] == '(' || res[i][j] == ')' ||
            res[i][j] == 'i' || res[i][j] == '$')
            {   install(prod[i], res[i][j]);
                break;
            }
            else if(res[i][j] == 'E' || res[i][j] == 'P'
            || res[i][j] == 'T')
            {   if(res[i][j-1] == '+' || res[i][j-1]
                == '*' || res[i][j-1] == '(' ||
                res[i][j-1] == ')')
                {   install(prod[i], res[i][j-1]);
                    break;
                }
            }
        }
    }
}
```

```
while (top >= 0)
{ pro = stack [top][0];
  re = stack [top][1];
  --top;
  for (i=0; 0<6; ++i)
  { for (j=2; j>0; --j)
    { if (res[i][0] == prod && res[i][0] != prod[i])
      { install (prod[i], re);
        break;
      }
      else if (res[i][0] != '\0')
      { break;
      }
    }
  }
}
for (i=0; i<18; ++i)
{ printf ("\n\t");
  for (j=0; j<3; ++j)
    printf ("%c \t", arr[i][j]);
}
getch();
clrscr();
printf("\n\n");
for (i=0; i<18; ++i)
{ if (pri != arr[i][0])
  { pri = arr[i][0];
    printf ("\n\t %c -> ", pri);
  }
```

Experiment :

Date

Page No.

```
if (arr[i][2] == 'T')
    printf ("%c", arr[i][1]);
}
getch();
}
```

Experiment : 07

Date
Page No.

- Aim :- To show all the operations of a stack
- Software Required :- C/C++ Compiler, Notepad
- Theory

Stack is a LIFO (last in first out) structur. It is an ordered list of the same type of elements. A stack is a linear list where all insertions and deletions are permitted only at one end of the list. When elements are added to stack it grow at one end. Similarly, when elements are deleted from a stack, it shrinks at the same end.

- Code

```
#include <stdio.h>
#include <proces.h>
#include <stdlib.h>
#define MAX 5
int top = -1, stack [MAX];
void push ();
void pop ();
void display ();
void main()
{  int ch;
   while (1){
   } printf ("\n *** Stack Menu *** ");
```

```
Enter your choice(1-4):1

Enter element to push:4

*** Stack Menu ***

1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):3

Stack is...
4
1

*** Stack Menu ***

1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):_
```

Experiment :

Date

Page No.

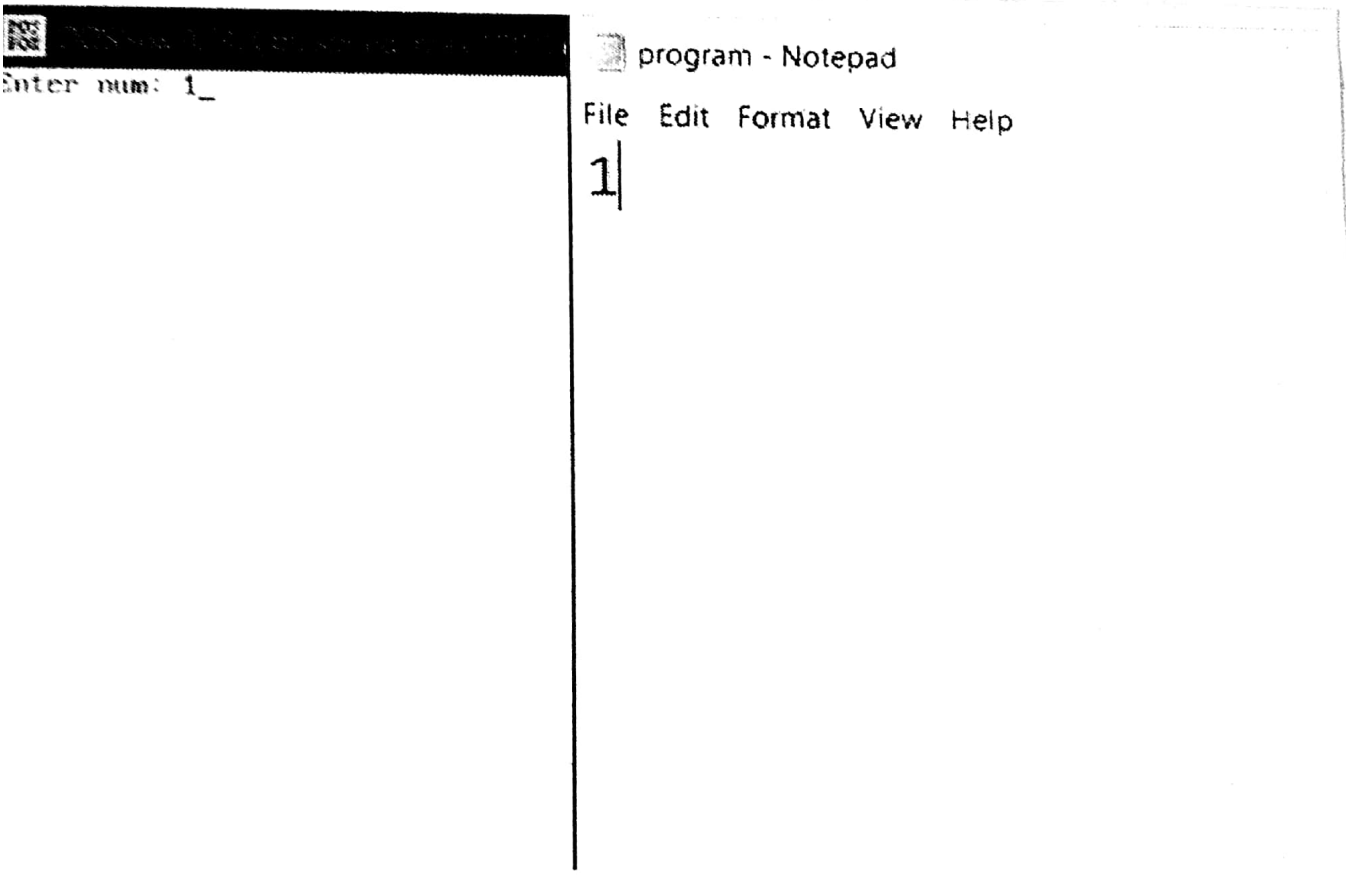
```
printf("\n\n 1. Push \n 2. Pop \n 3. Display \n
        4. Exit ");
printf("\n\n Enter your choice (1-4): ");
scanf("%d", &ch);
switch(ch)
{ case 1: push();
          break;
  case 2: pop();
          break;
  case 3: display();
          break;
  case 4: exit(0);
  default: printf("\n Wrong choice !! ");
  }
 }
}
void push()
{ int val;
  if (top == MAX-1)
  { printf("\n Stack is full !! ");
  }
  else
  { printf("\n Enter element to push: ");
    scanf("%d", &val);
    top = top+1;
    stack[top] = val;
  }
}
```

Experiment :

Date

Page No.

```
void pop()
{ if (top == -1)
  { printf("\n Stack is empty !! ");
  }
  else
  { printf("\n Deleted element is %d ", stack[top]);
    top = top-1;
  }
}
void display()
{ int i;
  if (top == -1)
  { printf ("\n Stack is empty !! ");
  }
  else
  { printf ("\n Stack is ....\n");
    for (i= top; i>0 ; --i)
        printf("%d \n", stack[i]);
  }
}
```



Date
Page No.

- Aim: Write a program to show various operations ie read, write & modify in a text file
- Software Required :- C/C++ Compiler, Notepad
- Theory

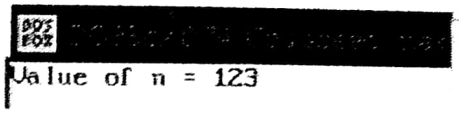
For reading and writing to a text file, we use the functions ~~fprintf() and~~ fprintf() & fscanf(). They are just the file versions of printf() and scanf(). They expects a pointer to the structure FILE. It takes the command from user & stores in the text file.

Function fread() & fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

- Code

– Write to a text file.

```
#include <stdio.h>
#include<stdlib.h>
Int main()
{ int num;
  FILE *fptr;
  fptr = fopen("c:\\program.txt", "w");
  if (fptr==NULL)
  { printf("Error!");
```



program - Notepad

File Edit Format View Help

123

```
    exit (1);
  }
  printf ("Enter num: ");
  scanf ("%d", &num);
  fprintf (fptr, "%d", num);
  fclose (fptr);
  return 0;
}
```

- Read from a text file

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{ int num;
  FILE *fptr;
  if ((fptr = fopen ("C:\\program.txt", "r")) == NULL)
  { printf ("Error! opening file");
    exit (1);
  }
  fscanf (fptr, "%d", &num);
  printf ("Value of n= %d", num);
  fclose (fptr);
  return 0;
}
```

- Modify a text file

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

Experiment :

Date

Page No.

```
int main()
{ FILE *ft;
  char ch;
  ft = fopen("C:\\program.txt", "r");
  if(ft==NULL)
  { printf("cannot open target file \n");
    exit(1);
  }
  while(1)
  { ch = fgetc(ft);
    if(ch=EOF)
    { printf("done")
      break;
    }
    if(ch=='i')
    { fputc('a', ft);
    }
  } fclose(ft);
    return 0;
}
```

Experiment : 09

Date
Page No.

- **Aim :-** Write a program to check whether a grammar is left recursive and remove left recursion.
- **Software Required :-** C/C++ compiler, notepad
- **Theory :-**

A production of grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS.

A grammar containing a production having left recursion is called as left recursion grammar

$A \rightarrow A\alpha / \beta$ (left recursive grammar)

Elimination of left recursion

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' / \epsilon$ (right recursive grammar)

- **Code**

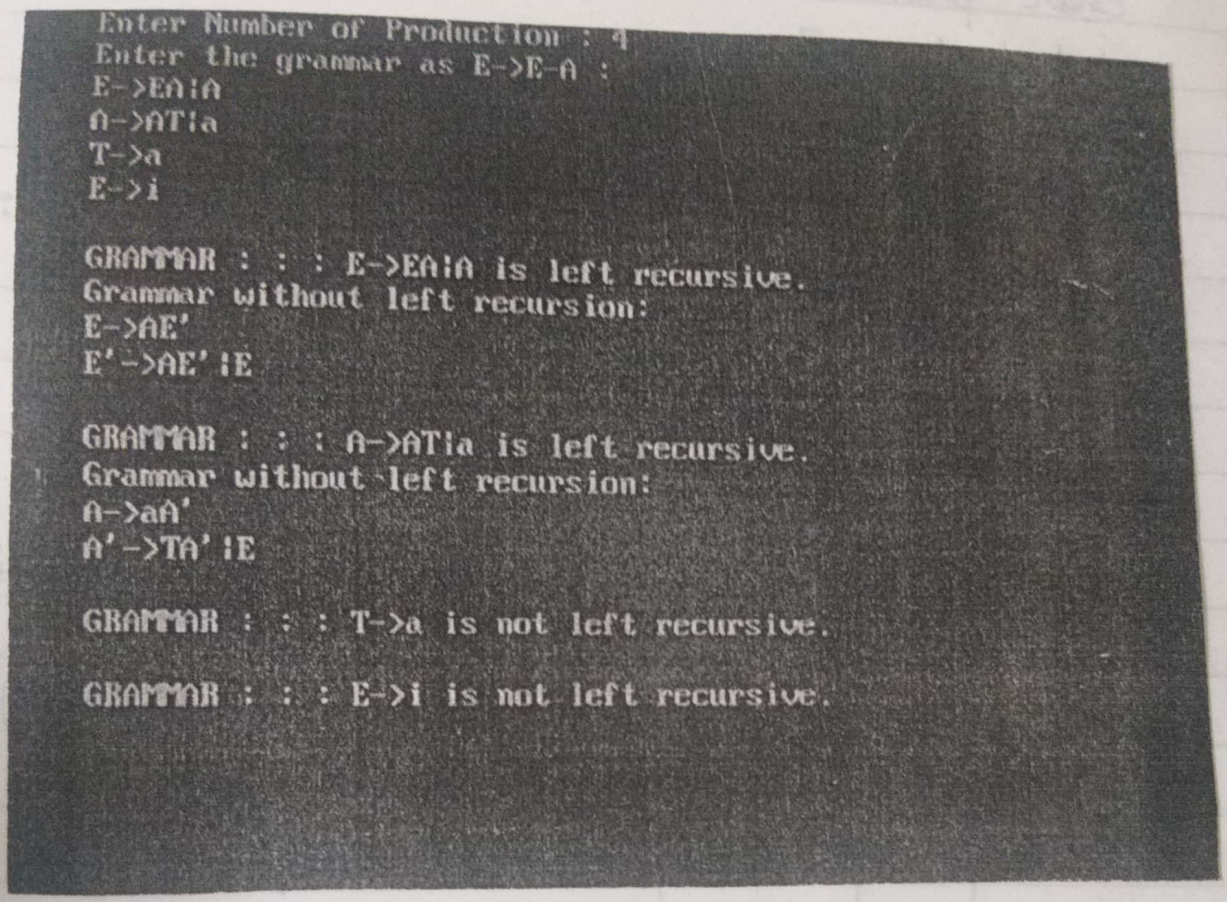
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define SIZE 10
int main()
{  clrscr();
   char non-terminal;
   char beta, alpha;
```

Experiment :

Date

Page No.

```
int num, i;
char production[10][SIZE];
int index = 3;
printf("Enter number of production :");
scanf("%d", &num);
printf("Enter the grammar as E->E-A :\n");
for(i=0; i<num; i++)
{  scanf("%s", production[i]);
}
for(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 w/o 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");
```



Experiment :

Date
Page No.

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

Experiment : 10

Date
Page No.

- **Aim** :- Write a program to remove left factoring
- **Software required** :- C/C++ compiler, notepad.
- **Theory**

In left factoring,

→ We make one production for each common prefixes

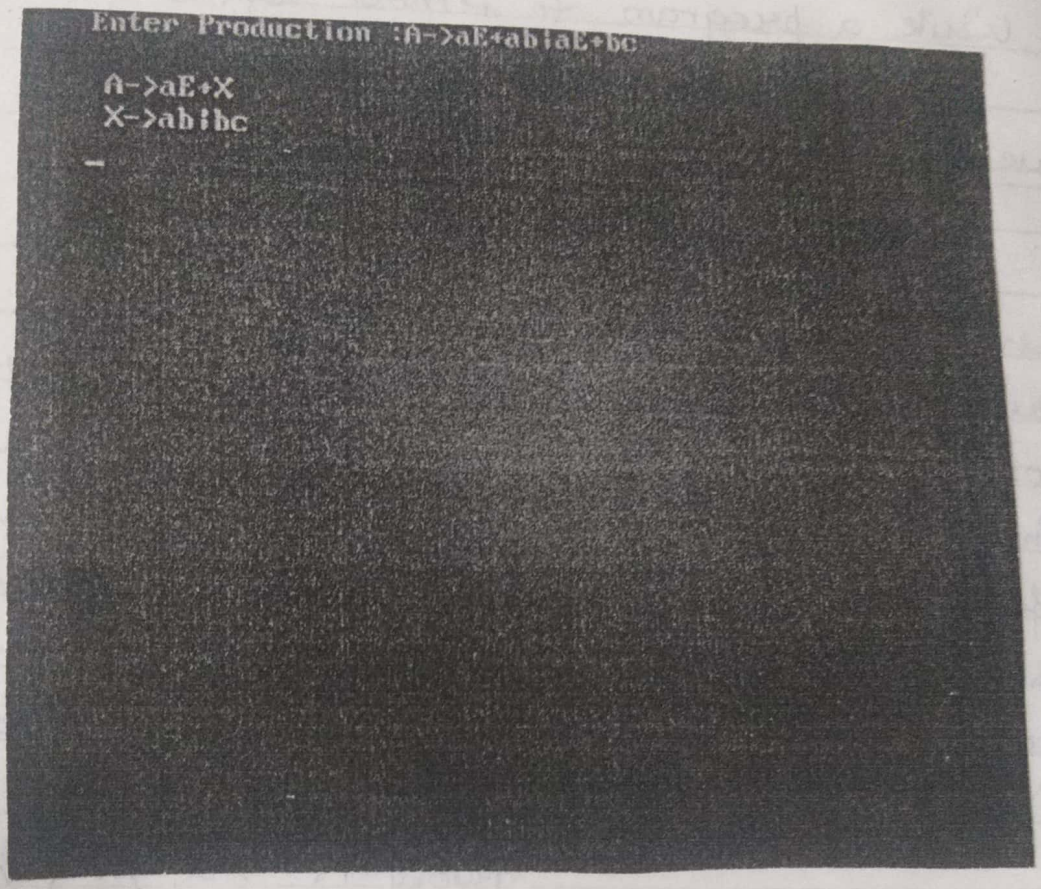
→ the common prefix may be a terminal or a non-terminal or a combination of both.

→ Rest of the derivation is added by new productions the grammar obtained after the process of left factoring is called as left factored grammar.

$A \to \alpha\beta_1 / \alpha\beta_2 / \alpha\beta_3 \xrightarrow[\text{factoring}]{\text{left}} A \to \alpha A'$, $A' \to \alpha_1 / \alpha_2 / \alpha_3$

- **Code**

```
#include <stdio.h>
#include<conio.h>
#include <string.h>
int main()
{ clrscr();
  char gram[20], part1[20], part2[20], modifiedGram[20]
  , newGram[20], tempGram[20];
  int i, j=0, k=0, l=0, pos;
  printf("Enter production : A->");
  gets(gram);
  for(i=0; gram[i]!='|'; i++, j++)
  { part1[j]=gram[i];
```



Experiment :

Date

Page No.

```
}
part1[j] = '\0';
for(j=++i; i=0; gram[j] != '\0'; j++, i++)
{ part2[i] = gram[j];
}
part2[i] = '\0';
for (i=0; i< strlen(part1) || i < strlen(part2); i++)
{ if(part1[i] == part2[i])
  { modifiedGram[k] = part1[i];
    k++;
    pos = i+1;
  }
}
for(i=pos, j=0; part1[i] != '\0'; i++, j++)
{ newGram[j] = part1[i];
}
newGram[j++] = '|';
for(i=pos; part2[i] != '\0'; i++, j++)
{ newGram[j] = part2[i];
}
modifiedGram[k] = 'X';
modifiedGram[++k] = '\0';
newGram[j] = '\0';
printf("\n A -> %s", modifiedGram);
printf("\n X -> %s\n", newGram);
getch();
}
```

Experiment : 11

Date

Page No.

- Aim :- Write a program to check whether a grammar is operator ~~procedure~~ precedent
- Software Requirement :- Turbo C/C++, notepad
- Theory :-
The computer science, an operator ~~procedure~~ precedence parser is a bottom up parser that interprets an operator precedence grammar
- Code :-

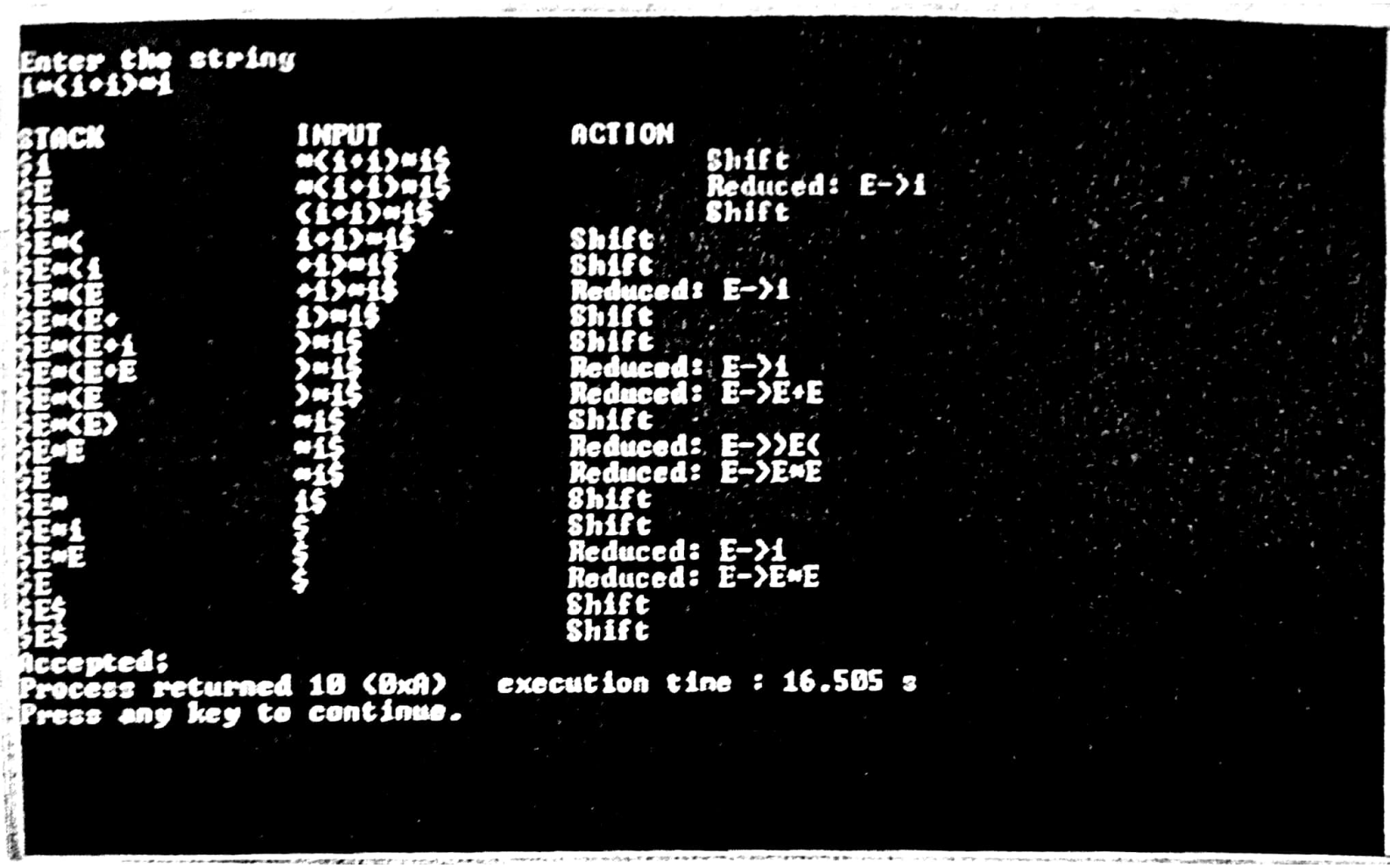
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char *input;
int i=0;
char lasthandle[6], stack[50], handle[][5]={"}[=
int to =0, 1;
char prec[9][9]= { /* input */
/*stack + - * / ^ i ( ) $ */
/* + */ '>', '>', '<', '<', ; */
int ~~getchar~~ getindex (char c)
{ switch (c)
  case '+': return 0;
  case '-': return 1;
  case '*': return 2;
  case '/': return 3;
  case '^': return 4;
```

Experiment :

Date

Page No.

```
case '1' : return 5;
case 'E' ; return 6;
case ')' : return 7;
case '$' : return 8;
}}
int shift()
{ stack[++top] = *(input ~~++i~~ +i++);
  stack[top+1] = '\0';
}
int reduce()
{ int i, len, found, t;
  for(i=0; i<5; i++)
  { len = strlen(handles[i]);
    if(stack[top] == handles[i][0] && top+1 >= len)
    { found=1;
      for(t=0; t<len; t++)
      { if(stack[top-t] != handles[i][t])
        { found=0;
          break;
        }}
      if(found==1)
      { stack[top-t+1] = 'E';
        top = top-t+1;
        strcpy(lasthandle, handles[i]);
        stack[top+1] = '\0';
        return 1;
      }
    }
  }
```



Experiment :

Date
Page No.

```
    return 0;
}
void dispstack()
{ int j;
for (j=0; j<=top; j++)
  printf("%c", stack[j]);
}
void dispinput()
{ int j;
  for (j=i; j<l; j++)
    printf("%c", stack[j]);
}
void main()
{ int j;
  input = (char *) malloc (50 * sizeof(char));
  printf("\n Enter the string \n");
  printf("\n Accepted");
  else
    printf("\n not accepted");
}
```

Experiment : 12

Date
Page No.

- Aim:- Write a program to generate a parse tree.
- Software required :- Turbo C/C++, Notepad.
- Theory:- Parse Tree is a hierarchical structure which represents the derivation of the grammar to yield ie., String. Root node of parse tree has the start symbol of the given grammar from where the derivation proceeds.

- Code :-

```
#include <stdio.h>
# Include <stdlib.h>
# Include <ctype.h>
char next;
void E (void) ; void T (void);
void S (void) ; void f (void);
void error ( int) ; void scan (void);
void enter (char);
void leave (char);
int level = 0;
Int main (void)
{ printf ("input");
  Scan (); E();
  if (next++ = '#') error ();
  else printf ("*** Succesful parser *** \n");
}
void E (void)
```

Experiment :

Date

Page No.

```
} enter ('E');
  T();
  while (next == '+' //next == '-')
  { scan();
    T();
  }
  leave('E');
}
void T(void)
} enter('T'); S();
  while(next == '*' // next == '/')
  } scan(); S();}
  leave('T');}
  void S(void)
  } enter('S');
    f();
  if (next == '^')
  } scan();
    S();
  leave('T');
  }
  void f(void)
  } enter('f'); if (isalpha(nex))
    }scan();
  { else if (next == '(').
  } scan(); f();
      if (next == ')')
      scan();
```

[redacted]

Scanned by CamScanner

Experiment:

Date

Page No.

```
else
}error (3);
}
leave ('f');
{
void scan (void)
} while (isspace (next=getchar ()));
{ void error (int n)
{ printf ("\n *** error \n", n);
exit (1);
}
void enter ( char name)
} spaces (level++);
{ void leave (char name)
{ spaces (--level)
printf ("t - %c \n", name);
}
void spaces (int local_level)
{ while (local_level--> 0)
printf ("|");
}
```