DATA STRUCTURES

By Omkar Deshpande

INDEX

Sr.	OBJECTIVES
11.	**Circular Linked List**
	a. Insert from Beginning, End and
	Intermediate.
	b. Delete from Beginning, End and
	Intermediate.
12.	**Doubly Linked List**
	a. Insert from Beginning, End and
	Intermediate.
	b. Delete from Beginning, End and
	Intermediate.
13.	**Linked Lists Application:**
	a. Addition of two given Polynomials
14.	**Trees:**
	a. Pre-Order
	b. In-Order
	c. Post-Order
15.	**Graphs:** Representations
	a. Adjacency Matrix
	b. Adjacency List

DATA STRUCTURES

The following file consists of all the important programs, of the subject Data Structures, implemented in the Practical Lab at MGM's College of Engineering, Nanded.

The following programs are implemented under the Guidance of **Bhagyashri Kapre Mam** and **Nikita Pande Mam.**

Note:

- All the C-Programs within this file, are stored at GitHub. GitHub is an online Code Management Platform.

Visit the following link to access all the C-Programs:
https://github.com/omkar98/SEMESTER-3

- For easy understanding, the explanation of all the programs (Practical Write-Up) along with Algorithms and C-Codes, can be accessed from the following blog:

http://engineersplot.blogspot.in/p/3rd-semester.html

(Go to "Data Structures" section in the blog, to access all the Practical Explanations of each and every program.)

Keep visiting the GitHub Directory, to get newly added programs or to get an updated version of previous programs. Also, the corresponding explanation (Practical Write-Up) of new programs would be uploaded on the blog "Engineers Plot" (engineersplot.blogspot.in).

Hence, stay updated by visiting these two websites regularly.

AIM: To find the area and perimeter of a rectangle using pointers.

(The main aim here is to print the values of area and perimeter from the main function)

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void area(int *, int *, int *, int *);
void main()
{
  int l,b, ar, peri;
  clrscr();
  printf("Enter length and breadth of
rectangle: ");
  scanf("%d %d", &l, &b);
  area(&l, &b, &ar, &peri);
  printf("\nIn Main function: \n Area =
%d; Perimeter = %d", ar, peri);
  getch();
}
```

```
  /* We use area and perimeter as *area
and *peri, because we want the values of
area and perimeter to be printed in
main function. Hence, we return the
addresses to main after calculation in
area function */
```

```
 void area(int *x, int *y, int *ar, int
*peri)
 {
   *ar=(*x)*(*y);
   *peri=(2*(*x) + 2*(*y));
   printf("\nIn area function: \nArea =
%d; Perimeter %d", *ar, *peri);
 }
```

```
==========OUTPUT=========
Enter length and breadth of rectangle: 3
4
In area function:
Area = 12; Perimeter 14
In Main function:
 Area = 12; Perimeter = 14
```

STRUCTURES

AIM: A simple program using structure that reads and displays details of a single book.

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void main()
{
  struct book
  {
  int b_id;
  char name[50];
  int price;
  } b;
  clrscr();
  printf("Enter Book name: ");
  gets(b.name);
//scanf cannot print the entire book
name, but only prints one word. Hence
we use gets.
  printf("Enter Book ID: ");
  scanf("%d", &b.b_id);
  printf("Enter Price: ");
  scanf("%d", &b.price);

  printf("\nYou have entered the
following book:\nBook ID: %d\nName:
%s\nPrice: Rs.%d\n",b.b_id, b.name,
b.price);
  getch();
}
```

```
========OUTPUT=======
Enter Book name: Programming in C
Enter Book ID: 203
Enter Price: 340

You have entered the following book:
Book ID: 203
Name: Programming in C
Price: Rs.340
```

AIM: A simple program using structure that reads and displays details of a single employee (using structures)

SYNTAX

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

void main()
{
struct emp
 {
  int id;
  char nm[50];
  char desg[50] ;
  long int sal;
 } e;
  clrscr();
  printf("\nEnter Employee ID and
Employee Name: ");
  scanf("%d", &e.id);
  gets(e.nm);
 // printf("Enter Employee name: ");
second printf()

 /*If we use printf before gets(as in this
case) then the program doesn't run
properly, i.e the compiler skips the next
printf() and gets() functions.
Problem: gets() only reads the input for
in the intial statement and skips in case
it is used for the next statements.
Solution: To read input using gets() for
subsequent statements, use gets()
directly after the scanf() statement.  */

  printf("Enter %s's Designation: ",
e.nm);
  scanf("%s", &e.desg);
  printf("Enter Salary: ");
  scanf("%ld", &e.sal);
  printf("ID \t
Name\t\tDesignation\tSalary");
  printf("\n%d.\t%s\t%s%ld", e.id,
e.nm, e.desg, e.sal);
  getch();  }
```

```
========OUTPUT=======
Enter Employee ID and Employee
Name: 203 Deshpande
Enter Deshpande's Designation:
Engineer
Enter Salary: 50000

ID      Name        Designation  Salary
203.   Deshpande  Engineer     Rs.50000
```

AIM: To add two numbers using structures.

SYNTAX

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

void main()
{
  clrscr();
  struct add
    {
     int a,b,c; // structure members
    }  s; // structure variable
  // struct add s;
  printf("Enter two numbers: ");
  scanf("%d %d", &s.a, &s.b);
  s.c=s.a+s.b;
  printf("Addition = %d", s.c);
  getch();
}
```

```
========OUTPUT=======
Enter two numbers: 2 3
Addition = 5
```

AIM: A program (using structures) that reads and displays details of as many books as user wants and later displays only those books, which lie within a certain range of user provided price.

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void main()
{  int i, n,max, min;
  clrscr();
  struct book
  {   int id, price;
   char name[50];
  } b[50]; //Let the max be 10 books.
  printf("Enter the number of Books: ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  {
   printf("Enter Book Name: ");
   scanf("%s",&b[i].name);
   printf("Enter Book ID and its price:
");
   scanf("%d %d", &b[i].id,
&b[i].price);
   }
   printf("\nID\tBook Name\tPrice\n");
   for(i=0; i<n; i++)
   {
   printf("%d.\t%s\tRs.%d\n",
b[i].id,b[i].name,b[i].price);
   }
   printf("\nEnter the range of price of
books, you are searching for: ");
   scanf("%d-%d", &min, &max);
   printf("\nFollowing books lie within
the range of Rs.%d to Rs.%d:
",min,max);
   printf("\nID\tBook Name\tPrice\n");
   for(i=0; i<n; i++)
   { if(b[i].price>=min &&
b[i].price<=max)
  {  printf("\n%d.\t%s\tRs.%d\n",
b[i].id,b[i].name,b[i].price);  }
   }
  getch(); }
```

========OUTPUT========

```
Enter the number of Books: 10
Enter Book Name: C_Language
Enter Book ID and its Price: 1 1900
Enter Book Name: Advanced_C
Enter Book ID and its Price: 2 2000
Enter Book Name: C++_Language
Enter Book ID and its Price: 3 2300
Enter Book Name: Advanced_C++
Enter Book ID and its Price: 4 2105
Enter Book Name: Web_Designing
Enter Book ID and its Price: 5 2500
Enter Book Name: Web_Develop
Enter Book ID and its Price: 6 970
Enter Book Name: JAVA_Language
Enter Book ID and its Price: 7 1700
Enter Book Name: Advanced_Java
Enter Book ID and its Price: 8 3001
Enter Book Name: HTML_Courses
Enter Book ID and its Price: 9 2999
Enter Book Name: Info_Tech
Enter Book ID and its Price: 10 3000

ID      Book Name        Price
1.     C_Language       Rs.1900
2.     Advanced_C       Rs.2000
3.     C++_Language     Rs.2300
4.     Advanced_C++     Rs.2105
5.     Web_Designing    Rs.2500
6.     Web_Develop      Rs.970
7.     JAVA_Language    Rs.1700
8.     Advanced_Java    Rs.3001
9.     HTML_Courses     Rs.2999
10.    Info_Tech        Rs.3000

Enter the range of price of books, you
are searching for: 2000-3000
Following books lie within the range of
Rs.2000 to Rs.3000:
ID      Book Name        Price
2.     Advanced_C       Rs.2000
3.     C++_Language     Rs.2300
4.     Advanced_C++     Rs.2105
5.     Web_Designing    Rs.2500
9.     HTML_Courses     Rs.2999
10.    Info_Tech        Rs.3000
```

Call by Value ← DIFFERENCE → Call by Reference

AIM : Perform swapping using regular fuctions (without using pointers).

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);

void main()
{
  clrscr();
  int a,b;
  printf("Enter two Numbers to swap:
");
  scanf("%d %d", &a, &b);
  printf("In main function = %d %d",
a,b);
  swap(a,b);
  getch();
}
void swap(int p, int q)
{
 int temp;
 temp=p;
 p=q;
 q=temp;
 printf("\nIn swap function = %d %d",
p,q);
/* We cannot use two return values
   return(p);
   return(q); */
}
```

```
=========OUTPUT========
Enter two Numbers to swap: 10 20
In main function = 10 20
In swap function = 20 10
```

- The changes made in the variables by the function is not reflected in the main function.
- Values of actual parameters (i.e a, b) are passed to the function.
- We can return only one value to the calling function.

AIM : Perform swapping using pointers.

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void swap(int *p , int *q);//With
Argument No Return
void main()
{
  clrscr();
  int a,b,d;
  printf("Enter two Numbers to swap:
");
  scanf("%d %d", &a, &b);
  swap(&a,&b);
  printf("\nIn main function = %d %d",
a,b);
  getch();
}

    // In swapping program
void swap(int *p, int *q)
{
 int temp;
 temp=*p;
 *p=*q;
 *q=temp;
 printf("In swap function = %d %d",
*p, *q);
}
```

```
=========OUTPUT==========
Enter two Numbers to swap: 10 20
In swap function = 20 10
In main function = 20 10
```

- The changes made in the variables by the function is reflected in the main function.
- The addresses (i.e &a &b) are passed to the function.
- We can return more than one values to the calling function (because we use pointers)

Name: Omkar Deshpande
Class: SE CSE 1
Roll No: 25
Subject: Data Structures

AIM:

- To insert an element in an array
- To delete an element from the array
- To modify(over-write) an element in the array

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int ary[30], ele, i,  num, pstn, ch;
 clrscr();
 printf("Note: Only 30 elements can be
inserted!\n");
 printf("Enter the number of elements you
want in the array: ");
 scanf("%d", &num);
/*Scans the elements*/
  for(i=0;i<num;i++)
  {
    scanf("\t%d", &ary[i]);
  }
  start: /*Goto statement*/
           printf("\nSelect an Operation to be
         performed:\n\t[1]
         INSERTION\n\t[2]
         DELETION\n\t[3] MODIFIY\n\t[4]
         EXIT");
           printf("\nYour Choice: ");
           scanf("%d", &ch);
/*A switch case is used to toggle between
various operations.*/
  switch(ch)
  {
           case 1: /*To insert an element*/
           printf("Enter the element you want
         to insert: ");
           scanf("\n%d", &ele);

           printf("Enter the position to insert
         the element: ");
           scanf("%d", &pstn);

           for(i=num;i>=pstn;i--)
           {
             ary[i]=ary[i-1];
           }
          num++;
          ary[pstn-1]=ele;
           for(i=0;i<num;i++)
           {
            printf("\t%d", ary[i]);  }
```

```
/* To go back to the previous Menu we use goto statement*/
          goto start;

  case 2:  /*To Delete an element*/
            printf("Enter the position of
          element you want to delete: ");
            scanf("%d", &pstn);
            for(i=pstn;i<num;i++)
            {
             ary[i-1]=ary[i];
            }
            num--;
            for(i=0;i<num;i++)
            {
             printf("\t%d", ary[i]);
            }
            goto start;

  case 3:  /*To Modify an Element*/
             printf("Enter the position of the
          element you want to modify: ");
             scanf("%d", &pstn);
             printf("Enter the element you
          want to insert in %d position: ",
          pstn);
             scanf("%d", &ele);
             ary[pstn-1]=ele;
             printf("New Array is: ");
             for(i=0;i<num;i++)
             {
              printf("\t%d", ary[i]);
             }
             goto start;

  case 4: break;
  }
          printf("\nProgram Terminated.");
          getch();
}
```

==========Output========

```
Note: Only 30 elements can be inserted!
Enter the number of elements you want in the array: 5
10
20
30
40
50
Select an Operation to be performed:
        [1] INSERTION
        [2] DELETION
        [3] MODIFIY
        [4] EXIT
Your Choice: 1
Enter the element you want to insert: 90
Enter the position to insert the element: 3
10      20      90      30      40      50
Select an Operation to be performed:
        [1] INSERTION
        [2] DELETION
        [3] MODIFIY
        [4] EXIT
Your Choice: 2
Enter the position of element you want to delete: 2
        10      90      30      40      50
Select an Operation to be performed:
        [1] INSERTION
        [2] DELETION
        [3] MODIFIY
        [4] EXIT
Your Choice: 3
Enter the position of the element you want to modify: 2
Enter the element you want to insert in 2 position: 20
New Array is:   10      20      30      40      50
Select an Operation to be performed:
        [1] INSERTION
        [2] DELETION
        [3] MODIFIY
        [4] EXIT
Your Choice: 4
Program Terminated.
```

AIM: A simple program using structure that reads and displays details of a single book.

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void main()
{
  struct book
  {
  int b_id;
  char name[50];
  int price;
  } b;
  clrscr();
  printf("Enter Book name: ");
  gets(b.name);
//scanf cannot print the entire book name, but only prints one word. Hence we use gets.
  printf("Enter Book ID: ");
  scanf("%d", &b.b_id);
  printf("Enter Price: ");
  scanf("%d", &b.price);

  printf("\nYou have entered the following book:\nBook ID: %d\nName: %s\nPrice: Rs.%d\n",b.b_id, b.name, b.price);
  getch();
}
```

```
========OUTPUT=======
Enter Book name: Programming in C
Enter Book ID: 203
Enter Price: 340

You have entered the following book:
Book ID: 203
Name: Programming in C
Price: Rs.340
```

AIM: A simple program using structure that reads and displays details of a single employee (using structures)

SYNTAX

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

void main()
{
struct emp
 {
  int id;
  char nm[50];
  char desg[50] ;
  long int sal;
 } e;
  clrscr();
  printf("\nEnter Employee ID and Employee Name: ");
  scanf("%d", &e.id);
  gets(e.nm);
  printf("Enter %s's Designation: ", e.nm);
  scanf("%s", &e.desg);
  printf("Enter Salary: ");
  scanf("%ld", &e.sal);
  printf("ID \t Name\t\tDesignation\tSalary");
  printf("\n%d.\t%s\t%s%ld", e.id, e.nm, e.desg, e.sal);
  getch();  }
```

```
========OUTPUT=======
Enter Employee ID and Employee Name: 203 Deshpande
Enter Deshpande's Designation: Engineer
Enter Salary: 50000

ID      Name        Designation  Salary
203.    Deshpande   Engineer     Rs.50000
```

AIM: A program (using structures) that reads and displays details of as many books as user wants and later displays only those books, which lie within a certain range of user provided price.

SYNTAX

```
#include<stdio.h>
#include<conio.h>
void main()
{  int i, n,max, min;
  clrscr();
  struct book
  {   int id, price;
   char name[50];
  } b[50]; //Let the max be 10 books.
  printf("Enter the number of Books: ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  {
   printf("Enter Book Name: ");
   scanf("%s",&b[i].name);
   printf("Enter Book ID and its price:
");
   scanf("%d %d", &b[i].id,
&b[i].price);
   }
   printf("\nID\tBook Name\tPrice\n");
   for(i=0; i<n; i++)
   {
   printf("%d.\t%s\tRs.%d\n",
b[i].id,b[i].name,b[i].price);
   }
   printf("\nEnter the range of price of
books, you are searching for: ");
   scanf("%d-%d", &min, &max);
   printf("\nFollowing books lie within
the range of Rs.%d to Rs.%d:
",min,max);
   printf("\nID\tBook Name\tPrice\n");
   for(i=0; i<n; i++)
   { if(b[i].price>=min &&
b[i].price<=max)
  {  printf("\n%d.\t%s\tRs.%d\n",
b[i].id,b[i].name,b[i].price);  }
   }
  getch(); }
```

========OUTPUT========

```
Enter the number of Books: 10
Enter Book Name: C_Language
Enter Book ID and its Price: 1 1900
Enter Book Name: Advanced_C
Enter Book ID and its Price: 2 2000
Enter Book Name: C++_Language
Enter Book ID and its Price: 3 2300
Enter Book Name: Advanced_C++
Enter Book ID and its Price: 4 2105
Enter Book Name: Web_Designing
Enter Book ID and its Price: 5 2500
Enter Book Name: Web_Develop
Enter Book ID and its Price: 6 970
Enter Book Name: JAVA_Language
Enter Book ID and its Price: 7 1700
Enter Book Name: Advanced_Java
Enter Book ID and its Price: 8 3001
Enter Book Name: HTML_Courses
Enter Book ID and its Price: 9 2999
Enter Book Name: Info_Tech
Enter Book ID and its Price: 10 3000

ID      Book Name        Price
1.      C_Language       Rs.1900
2.      Advanced_C       Rs.2000
3.      C++_Language     Rs.2300
4.      Advanced_C++     Rs.2105
5.      Web_Designing    Rs.2500
6.      Web_Develop      Rs.970
7.      JAVA_Language    Rs.1700
8.      Advanced_Java    Rs.3001
9.      HTML_Courses     Rs.2999
10.     Info_Tech        Rs.3000

Enter the range of price of books, you
are searching for: 2000-3000
Following books lie within the range of
Rs.2000 to Rs.3000:
ID      Book Name        Price
2.      Advanced_C       Rs.2000
3.      C++_Language     Rs.2300
4.      Advanced_C++     Rs.2105
5.      Web_Designing    Rs.2500
9.      HTML_Courses     Rs.2999
10.     Info_Tech        Rs.3000
```

AIM: To implement stack data structure and its Operations (i.e PUSH and POP)

SYNTAX:

```
#include<stdio.h>
#include<conio.h>
# define MAX 5
```

/* Drawback: Only 5 elements can be inserted at MAX in this array. We need to use dynamic memory allocation, in order to dynamically give size of array at the time of execution. */

/* **Global Declaration** of variables */

```
int a[4],ch,n,top=-1,ele,i;
```

/* **Push Function** pushes the element in stack. */

```
int push (int a[], int ele)
 {
  if(top==MAX-1)
    {
      printf("\nStack is full\n\n");
      return (1);
    }
  else
    {
       ++top;
       a[top]=ele;
       printf("%d is inserted\n", ele);
       printf("Elements in stack are:\n ");
       for(i=top;i>=0; i--)
        {
          printf("| %d |\n", a[i]);
        }
     return(0);
    }
 }
```

/* **Pop Function** pops out the top element in stack */

```
void pop (int a[])
 {
   int x;
    if(top==-1)
     {
      printf("\nStack is already
empty.\nPlease push an element first.\n");
     }
  else
    {
            x=a[top];
            --top;
            printf("%d is POPed out of
        stack.\n",x);
       if(top>=0)
       {
         printf("The element in the stack
are:");
              for(i=top;i>=0; i--)
                 {
                   printf("\n| %d |\n", a[i]);
                 }
     }
    }
 }

void main()
{
 clrscr();
```

/* We use **Goto statement** to return back to the main menu */

```
 start:
 printf("Select the operation you would like
to perform: ");
printf("\n\t[1]PUSH\n\t[2]POP\n\t[3]EXIT\
n");
 printf("Your Choice: ");
 scanf("%d", &ch);
 switch(ch)
 {
  case 1:
         printf("Enter the element you want
to push in the stack: ");
         scanf("%d", &ele);
         push(a,ele);
         goto start;

  case 2:
         pop(a);
         goto start;
```

```
 case 3:       break;

 default:
     printf("\nPlease enter a valid input");
 }
  printf("Program Terminated\n");
 getch();
}
```

==========**OUTPUT**==========

1. The user enters elements one by one in the stack.

Your Choice: 1
Enter the element you want to push in the stack: 33
33 is inserted
Elements in stack are:
| 33 |
| 20 |
| 22 |
| 45 |
| 23 |
Select the operation you would like to perform:
[1]PUSH
[2]POP
[3]EXIT
Your Choice: 1
Enter the element you want to push in the stack: 45

Stack is full

---

2. When the user wants to POP out all the elements from the stack.

Your Choice: 2
45 is POPed out of stack.
The element in the stack are:
| 23 |
Select the operation you would like to perform:
[1]PUSH
[2]POP
[3]EXIT
Your Choice: 2
23 is POPed out of stack.
Select the operation you would like to perform:
[1]PUSH
[2]POP
[3]EXIT
Your Choice: 2

Stack is already empty.
Please push an element first.

---

3. Hence, Push and Pop operation is done. The user decides to exit the program:

Select the operation you would like to perform:
[1]PUSH
[2]POP
[3]EXIT
Your Choice: 3
Program Terminated

Name: Omkar Deshpande
Class: SE CSE 1
Roll No: 25
Subject: Data Structures

AIM: To display operators according to their precedence.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j;
char stack[10];
char p[10]={'(',')','^','%','/','*','+','-'};
clrscr();
printf("Enter the operator: ");
for( i=0;i<=8;i++)
{
scanf("%c",&stack[i]);
}
for(i=0;i<=8;i++)
{
for(j=0;j<=8;j++)
{
if(p[i]==stack[j])
{
printf("%c\t",stack[j]);
}
}
}
getch();
}
```

==========**Output**========

```
Enter the operator: +-)(*^%/
(       )       ^       %       /       *       +
        -
```

AIM: To convert given Infix expression to Postfix.

PROGRAM:

```
/* This program converts infix expression to
postfix expression.
 * This program assume that there are Five
operators: (*, /, +, -,^)
        in infix expression and operands can
be of single-digit only.
 * This program will not work for fractional
numbers.
 * Further this program does not check
whether infix expression is
 valid or not in terms of number of operators
and operands.*/

#include<stdio.h>
#include<stdlib.h>      /* for exit() */
#include<ctype.h>     /* for isdigit(char ) */
#include<string.h>

#define SIZE 100

/* declared here as global variable because
stack[]
* is used by more than one fucntions */
char stack[SIZE];
int top = -1;

/* define push operation */

void push(char item)
{
        if(top >= SIZE-1)
        {
                printf("\nStack Overflow.");
        }
        else
        {
                top = top+1;
                stack[top] = item;
```

```
	}
}

/* define pop operation */
char pop()
{
	char item ;

	if(top == -1)
	{
		printf("stack under flow: invalid infix expression");
		getchar();
		/* underflow may occur for invalid expression */
		/* where ( and ) are not matched */
		exit(1);
	}
	else
	{
		item = stack[top];
		top = top-1;
		return(item);
	}
}

/* define function that is used to determine whether any symbol is operator or not
(that is symbol is operand)
* this fucntion returns 1 if symbol is opreator else return 0 */

int is_operator(char symbol)
{
	if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
	{
		return 1;
	}
	else
	{
	return 0;
	}
}

/* define fucntion that is used to assign precendence to operator.
* Here ^ denotes exponent operator.
* In this fucntion we assume that higher integer value
* means higher precendence */

int precedence(char symbol)
{
	if(symbol == '^')/* exponent operator, highest precedence*/
	{
		return(3);
	}
	else if(symbol == '*' || symbol == '/')
	{
		return(2);
	}
	else if(symbol == '+' || symbol == '-')	/* lowest precedence */
	{
		return(1);
	}
	else
	{
		return(0);
	}
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
	int i, j;
```

```
        char item;
        char x;

        push('(');                          /* push
'(' onto stack */
        strcat(infix_exp,")");              /*
add ')' to infix expression */

        i=0;
        j=0;
        item=infix_exp[i];          /* initialize
before loop*/

        while(item != '\0')         /* run loop
till end of infix expression */
        {
                if(item == '(')
                {
                        push(item);
                }
                else if( isdigit(item) ||
isalpha(item))
                {
                        postfix_exp[j] = item;
/* add operand symbol to postfix expr */
                        j++;
                }
                else if(is_operator(item) ==
1)       /* means symbol is operator */
                {
                        x=pop();
                        while(is_operator(x) ==
1 && precedence(x)>=
precedence(item))
                        {
                                postfix_exp[j] =
x;              /* so pop all higher
precendence operator and */
                                j++;
                                x = pop();
/* add them to postfix expresion */
                        }
                        push(x);
                        /* because just above
while loop will terminate we have
                        oppped one extra item
                        for which condition
fails and loop terminates, so that one*/

                        push(item);
/* push current oprerator symbol onto
stack */
                }
                else if(item == ')')        /* if
current symbol is ')' then */
                {
                        x = pop();                  /*
pop and keep popping until */
                        while(x != '(')
/* '(' encounterd */
                        {
                                postfix_exp[j] =
x;
                                j++;
                                x = pop();
                        }
                }
                else
                { /* if current symbol is
neither operand not '(' nor ')' and nor
                        operator */
                        printf("\nInvalid infix
Expression.\n");       /* the it is illegeal
symbol */
                        getchar();
                        exit(1);
                }
                i++;


                item = infix_exp[i]; /* go to
next symbol of infix expression */
```

```
	} /* while loop ends here */
	if(top>0)
	{
		printf("\nInvalid infix Expression.\n");	/* the it is illegeal symbol */
		getchar();
		exit(1);
	}
	if(top>0)
	{
		printf("\nInvalid infix Expression.\n");	/* the it is illegeal symbol */
		getchar();
		exit(1);
	}


	postfix_exp[j] = '\0'; /* add sentinel else puts() fucntion */
	/* will print entire postfix[] array upto SIZE */

}

/* main function begins */
int main()
{
	char infix[SIZE], postfix[SIZE]; /* declare infix string and postfix string */

	/* why we asked the user to enter infix expression
	* in parentheses ( )
	* What changes are required in porgram to
	* get rid of this restriction since it is not
	* in algorithm
	* */
	printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
	printf("\nEnter Infix expression : ");
	gets(infix);

	InfixToPostfix(infix,postfix); /* call to convert */
	printf("Postfix Expression: ");
	puts(postfix);			/* print postfix expression */

	return 0;
}
```

==========**OUTPUT**========

```
ASSUMPTION: The infix expression contains single letter variables and single dig
it constants only.
Enter Infix expression : (A+B)*C
Postfix Expression: AB+C*

ASSUMPTION: The infix expression contains single letter variables and single dig
it constants only.
Enter Infix expression : (A+B)
Postfix Expression: AB+
```

AIM: Write a program to evaluate a postfix expression

PROGRAM

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

/* Let the max value of the arrays we use be 100 */
#define MAX 100

/* Declare few global variables which can be used by more than 1 function */
int stack[MAX], top=-1;

/* We need the following operations to perform PostFix Evaluation
1) PUSH Function
2) POP Function
3) IsOperand Function
4) IsOperator Function
5) Operation Function (Optional, can also be included in the main())
*/

/* We don't consider the case to print stack overflow condition or underflow condition, as this will be printed in the output along with the Postfix expression.*/

/*------PUSH Function-----*/
void push(int value)
{
  ++top;
  stack[top]=value;
}
/*------POP Function------*/
int pop()
{
 int x = stack[top];
 --top;
 return(x);
}
/*------IsOperand Function---*/
/*This function returns 1(if operand) else 0*/
int isOperand(char symbol)
{
 if(symbol>='a' && symbol<='z')
    return 1;
 else
    return 0;
}
/*-----IsOperator Function ---*/
/*This Function returns 1(if operator) else returns 0*/
int isOperator(char symbol)
{
 if(symbol=='^' || symbol=='*' || symbol=='/' || symbol=='+' || symbol=='-')
    return 1;
 else
    return 0;
}
```

```
/*---------MAIN Function------*/
void main()
{
/*
STEP 1: Get the postfix expression
STEP 2: Evaluate each postfix symbol, if Operator or Operand
    2.1: If Operand, ask user for the value of operand
    2.2: 1: If Operator, pop first 2 elements from the postfix
      2.2.1: Perform the operation between the 2 operands
      2.2.2: Store it back into the stack
STEP 3: Repeat Step 2 till the end of Expression.
STEP 4: display final Result.
*/
  char postfix[MAX],symbol;
  int i=0, result, opr1, opr2, x, res_final;
  clrscr();
  printf("\nEnter the postfix Expression: ");
  gets(postfix);
  while(postfix[i]!='\0')
  {
    symbol=postfix[i];
    if(isOperand(symbol)==1)
    {
      printf("\nEnter the value of %c: ", symbol);
      scanf("%d", &x);
      push(x);
    }
    if(isOperator(symbol)==1)
    {
      opr2=pop();
      opr1=pop();
      switch(symbol)
      {
       case '^': result=pow(opr1,opr2);break;
       case '*': result=opr1*opr2;break;
       case '/': result=opr1/opr2;break;
       case '+':  result=opr1+opr2; break;
       case '-':  result=opr1-opr2; break;
      }
      push(result);
    }
    i++;//incrementing to check for another postfix symbol.
  }//While loop ending
  /*Once all the symbols are scanned, we pop the final result*/
   res_final=pop();
  printf("\nThe result of the Postfix Expression is: %d", res_final);
  getch();
}
```

OUTPUT

```
Enter the postfix Expression:
abc*+
Enter the value of a: 1
Enter the value of b: 2
Enter the value of c: 3
The result of the Postfix Expression is: 7
```

AIM: Write a program to check imbalance of parenthesis using stack.

To execute the program as required, we need to consider the following main points:

- Number of opening parenthesis('{') must be same as number of closing parenthesis('}').
- For every '}', there must be a corresponding '{' **before**.
- At any moment of time number of '{' must be >= number of '}'.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 50
/* Declare few global variables which can be used by more than 1 function */
char stack[MAX];
int top=-1;
/*-------PUSH Function------*/
void push(char sym)
{
    ++top;
    stack[top]=sym;
}
/*-------POP Function------*/
int pop()
{
  if(top==-1)
  {
   return 0;
  }
  else
  {
    stack[top];
    --top;
    return 1;
  }
}
/*-------Check Function------*/
void check(char exp[])
{
  int length;
  length = strlen(exp);
  for(int i=0; i<length; i++)
  {
```

```
   if(exp[i]=='{')
   {
     push(exp[i]);
   }
   if(exp[i]=='}')
   {
     int x;
     x=pop();
     if(x==1)
     {
       printf("A closing bracket(at
position %d) has a balancing
opening bracket.\n", i+1);
     }
     if(x==0)
     {
       printf("There is no opening
bracket before a closing bracket
(of Postion-%d).\n",i+1);
     }
   }
  }/*For loop Termination*/
  if (pop()==1)
  {
    printf("There is/are extra
opening bracket(s).\n")   ;
  }
}
/*-------MAIN Function------*/
void main()
{
  char exp[MAX];
  clrscr();
  printf("Enter the expression: ");
  gets(exp);
  check(exp);
  getch(); }
```

OUTPUT

Enter the expression:
{a+b{c/d}}
A closing bracket(at position 9) has a balancing opening bracket.
A closing bracket(at position 10) has a balancing opening bracket.

Enter the expression:
{a+b{c/d}}*d}
A closing bracket(at position 9) has a balancing opening bracket.
A closing bracket(at position 10) has a balancing opening bracket.
There is no opening bracket for a closing bracket(of Postion-13).

Enter the expression:
{d*{a+b{c/d}}
A closing bracket(at position 12) has a balancing opening bracket.
A closing bracket(at position 13) has a balancing opening bracket.
There is/are extra opening bracket(s).

Enter the expression: **}}a+b{{**
There is no opening bracket before a closing bracket(of Postion-1).
There is no opening bracket before a closing bracket(of Postion-2).
There is/are extra opening bracket(s).

AIM: To find the sum of Fibonacci series using recursion function.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
/*Function Prototype*/
int  rec(int,int,int);

/*Since, sum variable is used by other function, hence we declare it globally.*/
int sum=0;

/*----Main Function----*/
void main()
{
	int n,m,k ;
	clrscr();
	printf("Enter the number: ");
	scanf("%d",&k);
	printf("\nSum: ");
	m=0;
	n=1;
	rec(m,n,k);
	getch();
}

/*RECURSION FUNCTION*/
int rec(int f1,int f2,int n)
{
	sum=f1+f2;
	if(sum<n)
	{
		printf(" %d ", sum);
		rec(f2,sum,n);
	}
}
```

OUTPUT

```
Enter the number: 10
Sum:  1  2  3  5  8
```

AIM: Implement Binary Search using Iteration.

PROGRAM

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

/*We have directly initialized the array with 10 elements placed in a sorted way (Ascending) */
int array[]={10,20,30,40,50,60,70,80,90,100};

/*---BINARY SEARCH -----*/
void search(int n, int first, int mid, int last)
{
  while(array[mid]!=n && first<=last)
   {
   mid=(first+last)/2;
   if(array[mid]==n)
    {
      printf("The given element is found at position %d.\n", mid+1);
      exit(0);
    }
else if(array[mid]<n)
    {
      mid++;
      first=mid;
    }
    else if(array[mid]>n)
    {
      mid--;
      last=mid;
    }
    else
    {
      printf("Element is not found in the list\n");
      exit(0);
    }
  } /*While loop termination*/

  if(array[mid]==n)
  printf("The element is found at position: %d", mid+1);
  else
  {
   printf("The element %d doesnot exist in the list.\n", n);
  }
} /*Function Terminates Here*/
```

```
/*------MAIN FUNCTION-----*/
void main()
{
 int n,i,first=0,last=9,mid;
 clrscr();

/*We can also get input from the user, instead of initializing the array.*/

 /*
printf("Enter the 10 numbers [Ascending order only]:\n ");
 for(i=0;i<10;i++)
  {
    scanf("%d", &array[i]);
  }
*/

 printf("\nEnter the element you want to search for: ");
 scanf("%d", &n);
 mid=(first+last)/2;
 search(n,first,mid,last);
  getch();
}
```

OUTPUT

```
Enter the element you want to search for: 60
The given element is found at position 6.
```

```
Enter the element you want to search for: 100
The element is found at position: 10
```

```
Enter the element you want to search for: 53
The element 53 doesnot exist in the list.
```

AIM: To implement Binary Search using Recursion.

PROGRAM

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

/*We have directly initialized the array with 10 elements placed in a sorted way (Ascending) */
int array[]={10,20,30,40,50,60,70,80,90,100};

/*---BINARY SEARCH -----*/
void search(int n, int first, int mid, int last)
{
 if(array[mid]!=n && first<=last)
 {
  mid=(first+last)/2;
  if(array[mid]==n)
   {
     printf("The given element is found at position %d.\n", mid+1);
     exit(0);
   }
  else if(array[mid]<n)
   {
     /*mid++;
     first=mid; (in case of iteration)*/
     search(n,mid+1,mid+1,last);
   }
   else if(array[mid]>n)
   {
     /*mid--;
     last=mid; (In case of iteration)*/
     search(n,first,mid-1,mid-1);
   }
   else
   {
     printf("Element is not found in the list\n");
     exit(0);
   }
 }
 if(array[mid]==n)
  {
   printf("The element is found at position: %d", mid+1);
   exit(0);
  }
  else
  {
   printf("The element %d doesnot exixt in the list.\n", n);
   exit(0);
  }
}
```

```
/*--------Main Function--------*/
void main()
{
 int n,i,first=0,last=9,mid;
 clrscr();

/*We can also get input from the user, instead of initializing the array.*/

 /*
printf("Enter the 10 numbers [Ascending order only]:\n ");
 for(i=0;i<10;i++)
  {
    scanf("%d", &array[i]);
  }
*/

 printf("\nEnter the element you want to search for: ");
 scanf("%d", &n);
 mid=(first+last)/2;
 search(n,first,mid,last);
  getch();
}
```

OUTPUT

```
Enter the element you want to search for: 60
The given element is found at position 6.
```

```
Enter the element you want to search for: 100
The element is found at position: 10
```

```
Enter the element you want to search for: 53
The element 53 doesnot exixt in the list.
```

AIM: Implement the program of Tower of Hanoi using recursion.

PROGRAM

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

int i=0;/*To set a counter, that counts the number of moves taken.*/

/*--Tower of Hanoi Function--*/
void TOH(int n, char beg, char end, char aux)
{
  if(n==1)
  {
    printf("Move No: %d  | Moved the disk 1 from %c to %c\n",++i,beg, end);
  }
  else
  {
    TOH(n-1,beg,aux,end);
    printf("Move No: %d  | Moved the disk %d from %c to %c\n",++i,n,beg,end);
    TOH(n-1,aux,end,beg);
  }
}

/*--------Main Function------*/
void main()
{
  int n;
  char beg='A',end='C',aux='B';
  clrscr();
  printf("Enter no. of Disks: ");
  scanf("%d", &n);
  TOH(n,beg,end,aux);
  getch();
}
```

OUTPUT

```
Enter no. of Disks: 3
Move No: 1  | Moved the disk 1 from A to C
Move No: 2  | Moved the disk 2 from A to B
Move No: 3  | Moved the disk 1 from C to B
Move No: 4  | Moved the disk 3 from A to C
Move No: 5  | Moved the disk 1 from B to A
Move No: 6  | Moved the disk 2 from B to C
Move No: 7  | Moved the disk 1 from A to C
```

OUTPUT

```
Enter no. of Disks: 4
Move No: 1  | Moved the disk 1 from A to B
Move No: 2  | Moved the disk 2 from A to C
Move No: 3  | Moved the disk 1 from B to C
Move No: 4  | Moved the disk 3 from A to B
Move No: 5  | Moved the disk 1 from C to A
Move No: 6  | Moved the disk 2 from C to B
Move No: 7  | Moved the disk 1 from A to B
Move No: 8  | Moved the disk 4 from A to C
Move No: 9  | Moved the disk 1 from B to C
Move No: 10  | Moved the disk 2 from B to A
Move No: 11  | Moved the disk 1 from C to A
Move No: 12  | Moved the disk 3 from B to C
Move No: 13  | Moved the disk 1 from A to B
Move No: 14  | Moved the disk 2 from A to C
Move No: 15  | Moved the disk 1 from B to C
```

AIM: To implement simple Queue and the following queue operations:

[1]. Enqueue
[2]. Dequeue
[3]. Display

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>/*for exit(0)*/

#define MAX 5

/*Globally declared variables, because these are used by more than one functions.*/
int num, A[MAX], rear=-1, front=-1;

/*-------Enqueue Function------*/
void enqueue(int num)
{
```

/*Following cases should be considered:
(i) An Overflow condition
(ii) Set the front to 0 from reset condition of -1and then insert the element.
***/**

```
  if(rear==MAX-1)
  printf("Overflow Condition\n");
  else
  {
    if(front==-1)
       front=0;

    ++rear;
    A[rear]=num;
    printf("Enqueued | %d |\n", num);
  }
}

/*-------Dequeue Function------*/
void dequeue()
{
```

/*
Following cases should be considered for dequeue-ing:
(i) If Front and Rear, both are -1 then the queue is empty.
(ii) Front cannot exceed Rear. Hence, the max condition for front should be front<rear.
(iii) If front=rear, then the element will be dequeued and front & rear gets reset to -1.
***/**

```
 if(front==-1 && rear==-1)
  {
    printf("UnderFlow
Condition.\n");
  }
  else if(front<rear)
  {
       int  temp = A[front];
       ++front;
       printf("Successfully
Dequeued | %d |.\n", temp);
  }
  else if(front==rear)
  {
       temp = A[front];
       printf("Successfully
Dequeued | %d |.\n", temp);
      front=-1;
      rear=-1;
  }
}
/*---------Display Function------*/
void display()
{
 int i;
 printf("The Elements in the queue
are: \n");
  for(i=front;i<=rear;i++)
  {
    printf(" %d |", A[i]);
  }
}
```

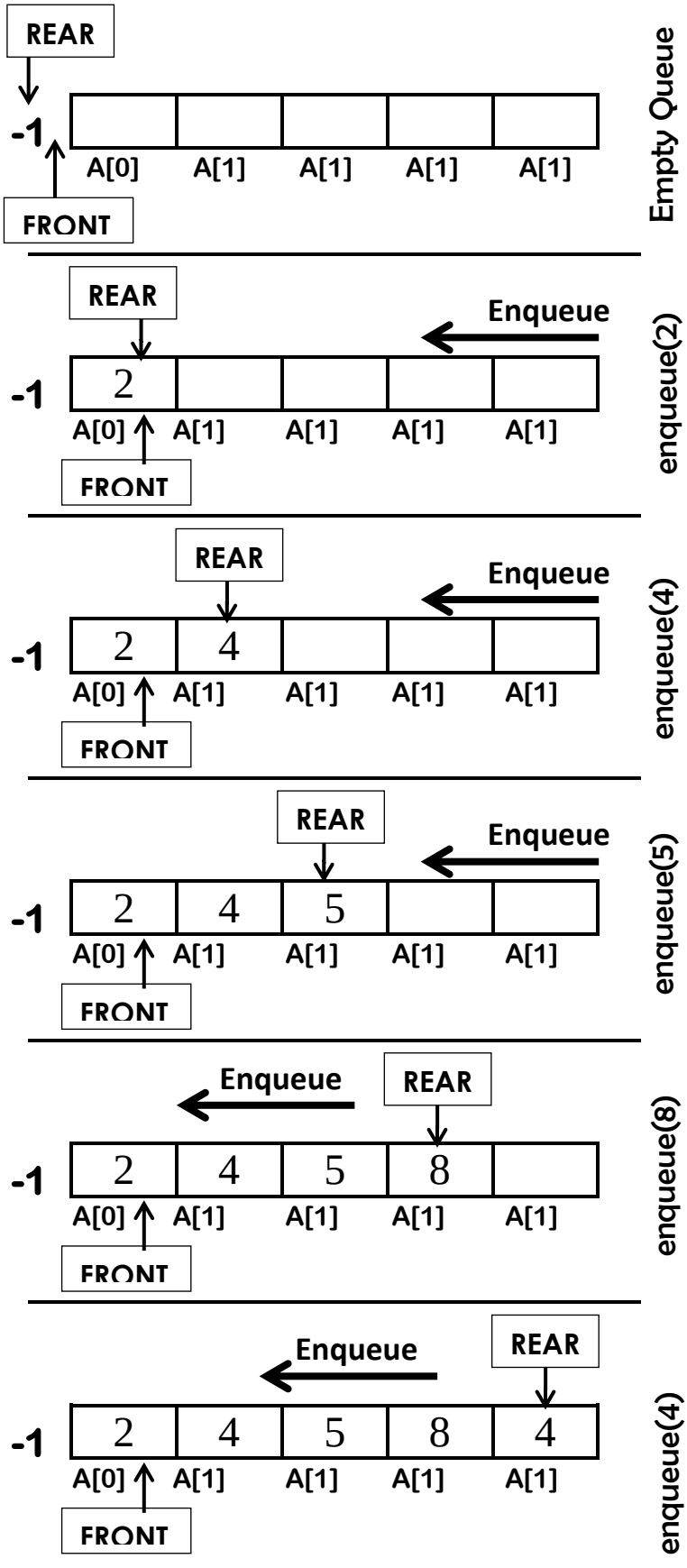
/*We display elements that lie within front to rear*/

```
void main()
{
  int ch;
  clrscr();
  printf("=====MENU====\n") ;
  printf("\t[1] Enqueue\n\t[2]
Dequeue\n\t[3] Display\n\t[4]
EXIT\n");
  start:  /*goto concept*/
  printf("\nEnter your choice: ");
  scanf("%d", &ch);
  switch(ch)
  {
    case 1:
       printf("Enter a number to
enqueue: ");
       scanf("%d", &num);
       enqueue(num);
       goto start;
    case 2:
        dequeue();
        goto start;
    case 3:
        display();
        goto start;
    case 4:
      exit(0);
    default:
         printf("Please enter a valid
choice.\n");
         goto start;
  }
  getch();
}
```

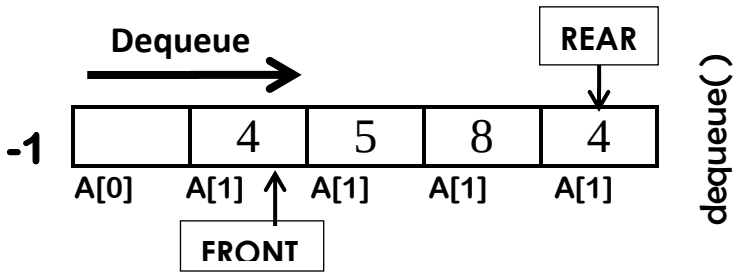
OUTPUT

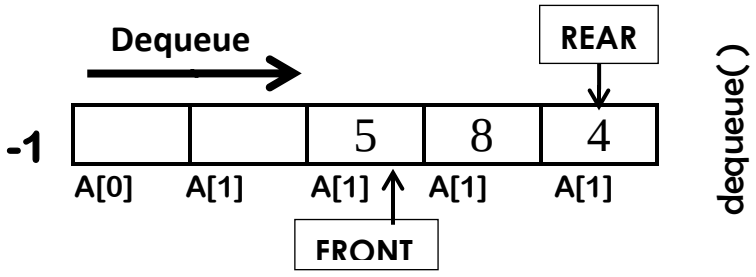
```
=====MENU====
      [1] Enqueue
      [2] Dequeue
      [3] Display
      [4] EXIT
Enter your choice: 1
Enter a number to enqueue: 2
Enqueued | 2 |

Enter your choice: 1
Enter a number to enqueue: 4
Enqueued | 4 |

Enter your choice: 1
Enter a number to enqueue: 5
Enqueued | 5 |

Enter your choice: 1
Enter a number to enqueue: 8
Enqueued | 8 |

Enter your choice: 1
Enter a number to enqueue: 4
Enqueued | 4 |

Enter your choice: 1
Enter a number to enqueue: 9
Overflow Condition
Enter your choice: 3
The Elements in the queue are:
2 | 4 | 5 | 8 | 4 |
```

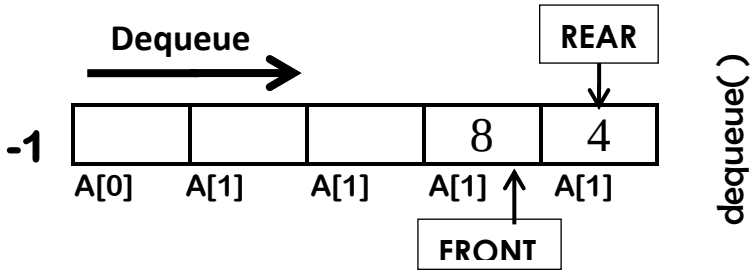
Diagram Representation



```
Enter your choice: 2
Successfully Dequeued | 2 |.

Enter your choice: 2
Successfully Dequeued | 4 |.

Enter your choice: 2
Successfully Dequeued | 5 |.

Enter your choice: 3
The Elements in the queue are:
| 8 | 4
```

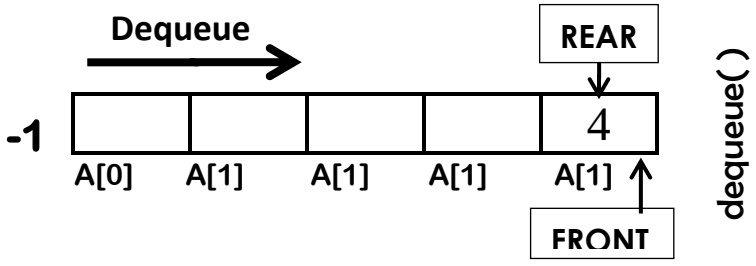




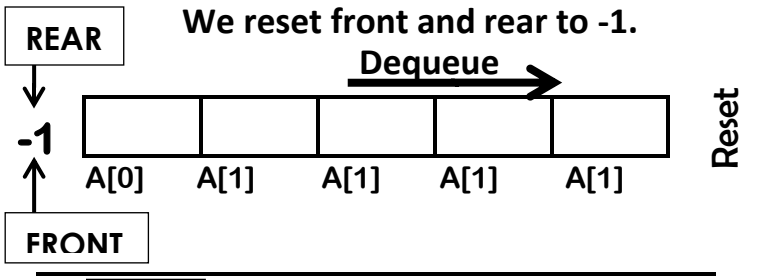


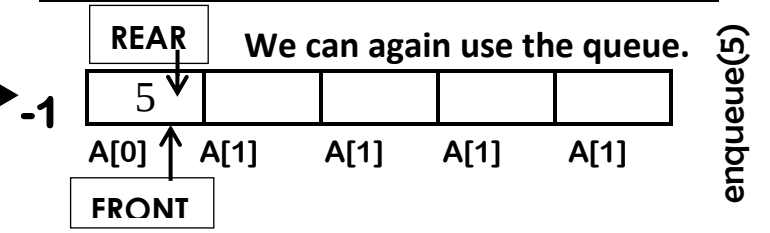

```
Enter your choice: 1
Enter a number to enqueue: 5
Overflow Condition

Enter your choice: 2
Successfully Dequeued | 8 |.
Enter your choice: 2
Successfully Dequeued | 4 |.


Enter your choice: 2
UnderFlow Condition.
```

Disadvantage: In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we cannot insert the next element even if there is a space in front of queue.




```
Enter your choice: 1
Enter a number to enqueue: 5
Enqueued | 5 |

Enter your choice: 4
```





AIM: Double Ended Queue using array implemention.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define max 10
 int queue[max],f=max/2,r=max/2;
 void enqueue_f();
 void enqueue_r();
 void dequeue_f();
 void dequeue_r();
 void display();
 void main()
{
 int ch;
 clrscr();
 printf("\n--- Double ended Queue
implemention ---");
 while(ch!=0)
 {
 printf("\n1.Enqueue_F   2.Enqueue_R
3.Dequeue_F 4.Dequeue_R 5.Display
0.Exit");
 printf("\nEnter a choice: ");
 scanf("%d",&ch);
 switch(ch)
{
 case 1: enqueue_f();
 break;
 case 2: enqueue_r();
 break;
 case 3: dequeue_f();
 break;
 case 4: dequeue_r();
 break;
 case 5: display();
 break;
 case 0: exit(0);
 break;
 default:
 printf("\nInvalid Choice");
 break;
 }
}
 getch();
}
      /*Enqueue from rear Operation*/
  void enqueue_r( int x)
{
  if(r==max)
   printf("\nCan not enqueue. ");
  else
   printf("\nEnter element to enqueue: ");
   scanf("%d",&x);
   queue[r]=x;
   r++;
}
      /*Enqueue from front Operation*/
  void enqueue_f( int x)
{
   if(f==-1)
   printf("\nCan not enqueue. ");
  else
   printf("\nEnter element to enqueue: ");
   scanf("%d",&x);
   f--;
   queue[f]=x;

}

      /*Dequeue from front Operation*/
  void dequeue_f()
{
  int t;
  if(f==max/2)
 {
  printf("\nQueue is underflow ");
 }
  else
 {
  t=queue[f];
  f++;
  printf("\n<%d> element is dequeued.",t);
 }

}
```

```
        /*Dequeue from rear Operation*/
void dequeue_r()
{
   int t;
   if(r==max/2)
  {
   printf("\nQueue is underflow from rear.");
  }
   else
  {
   t=queue[r-1];
   r-- ;
   printf("\n%d Element is dequeued from
rear.",t);

  }
}
                /*Display Function*/
   void display()
{
   int i;
   if(f==r)
  {
   printf("\nQueue is empty.");
  }
   else
  {
   printf("\nDisplaying elements in
queue:\n");
   printf("\n");
   for(i=f;i<r;i++)
  {
     printf("%d ",queue[i]);
  }

  }
}
```

OUTPUT

```
--- Double ended Queue implemention ---
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 1
Enter element to enqueue: 2

1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 1
Enter element to enqueue: 3

1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 1
Enter element to enqueue: 4

1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 5
Displaying elements in queue:
4 3 2
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 2
Enter element to enqueue: 34
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 2
Enter element to enqueue: 45
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 5
Displaying elements in queue:
4 3 2 34 45
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 3
<4> element is dequeued.
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 4
45 Element is dequeued from rear.
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 5
Displaying elements in queue:
3 2 34
1.Enqueue_F   2.Enqueue_R 3.Dequeue_F
4.Dequeue_R 5.Display 0.Exit
Enter a choice: 0       */
```

AIM: To implement a program of Circular Queue.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>/*For exit( )*/
#define MAX 5

/*We declare these variables globally, because they are used by more than one functions.*/
int queue[MAX], rear=-1,front=-1;
```

/* We need to consider all the following cases for enqueue:

Case 1: Overflow Conditions, i.e when **rear reaches MAX-1** and when **front exceeds rear by 1**.

Case 2: If **front is -1**(Reset Condition) then **set front and rear to 0.**

Case 3: When rear reaches MAX-1 but there is **empty space in front of queue**. Hence, we **set rear** to 0.

Case 4: It will be a **default** case. Here rear is incremented by 1 and the element gets into the queue.
*/

```
/*-------Enqueue Function------*/
int enqueue(int ele)
{
  if(rear==MAX-1 && front==0)
  {
    printf("QUEUE OVERFLOW.\n");
    return 0;
  }
  else if(front==rear+1)
  {
    printf("QUEUE OVERFLOW.\n");
    return 0;
  }
  else if(front==-1) /*Case 2*/
  {
    front=0;
    rear=0;
    queue[rear]=ele;
    return 1;
  }
  else if(rear==MAX-1 && front!=0)    /*Case 3*/
  {
    rear=0;
    queue[rear]=ele;
    return 1;
  }
  else
  {
    ++rear;
    queue[rear]=ele;
```

```
   return 1;
  }
}
```

/* We need to consider all the following cases for dequeue:

Case 1: Underflow Condition, i.e when **front and rear at -1 (Reset values)**.

Case 2: When queue will just be empty, i.e both front and rear are equal.

We reset them back to -1, after dequeue-ing the last remaining element.

Case 3: When front reaches MAX-1 but there are **elements in front of queue**. Hence, we **set front** to 0.

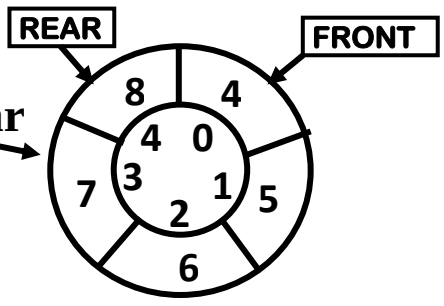
Case 4: It will be a **default** case. Here element will be stored in a variable and front is incremented by 1.

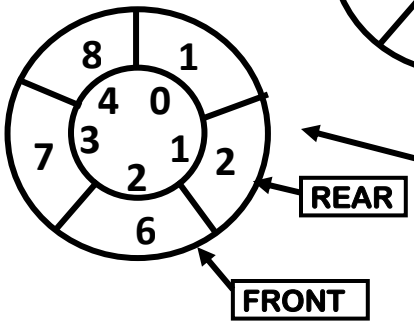
```
/*-------Dequeue Function------*/

int dequeue()
{
 int ele;
 if(front==-1 && rear==-1)/*Case-1*/
 {
   printf("UNDERFLOW CONDITION.\n");
   return 0;
 }
 else if(front==rear)/*Case-2*/
 {
   ele=queue[front];
   printf("Dequeued %d.", queue[front]);
   front=rear=-1;
   return ele;
 }
 else if(front==MAX-1)/*Case-3*/
 {
   ele=queue[front];
   printf("Dequeued %d.", queue[front]);
   front=0;
   return ele;
 }

else/*Case-4*/
 {
  ele=queue[front];
  printf("Dequeued %d.\n", queue[front]);
  ++front;
  return ele;
 }
}
```

```
/*--------Display Function------*/
void display()
{
int i;
 if(front<=rear)
 {
  for(i=front;i<=rear;i++)
  {
    printf("| %d |\t", queue[i]);
  }
 }
 if(front>rear)
 {
  for(i=front;i<MAX;i++)
    printf("| %d |\t", queue[i]);
  for(i=0;i<=rear;i++)
    printf("| %d |\t", queue[i]);
 }
}
```

Front can either be greater than or smaller than Rear. Hence we apply two conditions to display the elements in the queue.

Front <=Rear





Front>Rear

REAR

FRONT

```
/*--------MAIN Function-------*/
void main()
{
  int ele, ret=1, ch;
  clrscr();
  printf("=====MENU====\n");
  printf("\t[1] ENQUEUE\n\t[2]
DEQUEUE\n\t[3]
DISPLAY\n\t[4] EXIT\n");
  /*while(ret==1)
  {*/
   start:
   printf("\nYour choice: ");
   scanf("%d", &ch);
   switch(ch)
   {
    case 1:
      printf("Enter the element to
push: ");
      scanf("%d", &ele);
      ret=enqueue(ele);
      goto start;
    case 2:
      ret=dequeue();
      goto start;
    case 3:
      display();
      goto start;
    case 4:
      exit(0);
   }
 /* } while-loop termination*/
  getch();
}
```

We can either use while loop or goto statement.

OUTPUT

TEST CASE-1: Overflow - 1

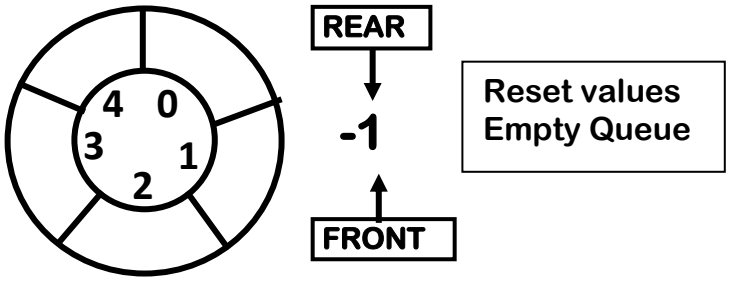
```
=====MENU====
      [1] ENQUEUE
      [2] DEQUEUE
      [3] DISPLAY
      [4] EXIT
Your choice: 1
Enter the element to push: 4
Your choice: 1
Enter the element to push: 5
Your choice: 1
Enter the element to push: 6
Your choice: 1
Enter the element to push: 7
Your choice: 1
Enter the element to push: 8
Your choice: 1
Enter the element to push: 9
QUEUE OVERFLOW.
```

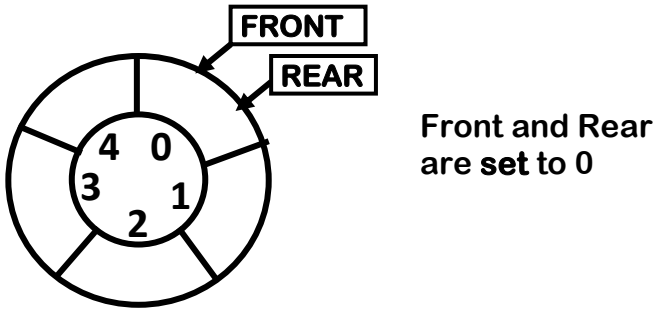
```
Your choice: 3
| 4 |  | 5 |  | 6 |  | 7 |  | 8 |
```

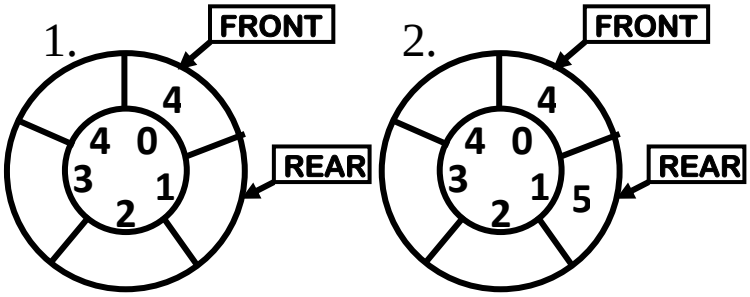
```
Your choice: 4
```

DIAGRAMATIC REPRESENTATION

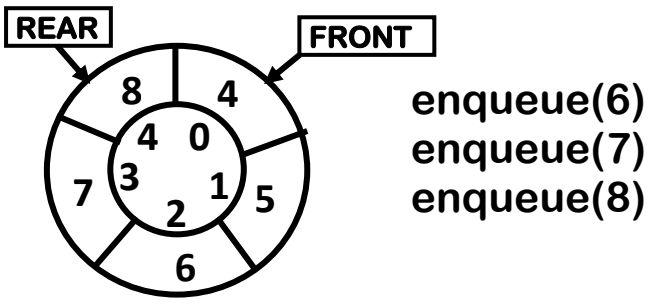
TEST CASE – 1: Overflow - 1







Rear is first incremented, ++rear;
and then the element is inserted into the queue, queue[rear]=ele;



TEST CASE-2: Overflow-2

Your choice: 2
Dequeued 4.
Your choice: 2
Dequeued 5.

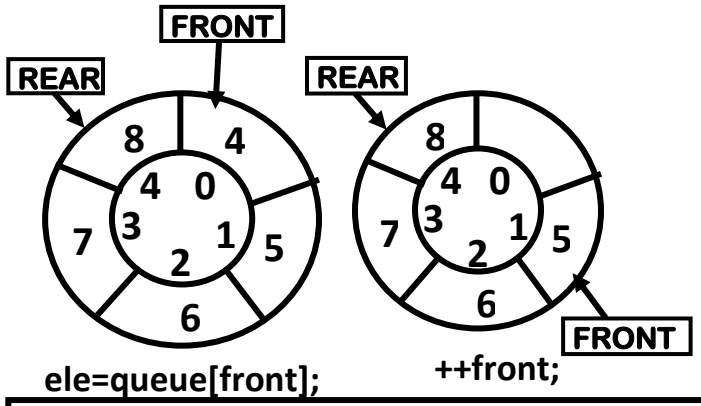
Your choice: 1
Enter the element to push: 1

Your choice: 1
Enter the element to push: 2

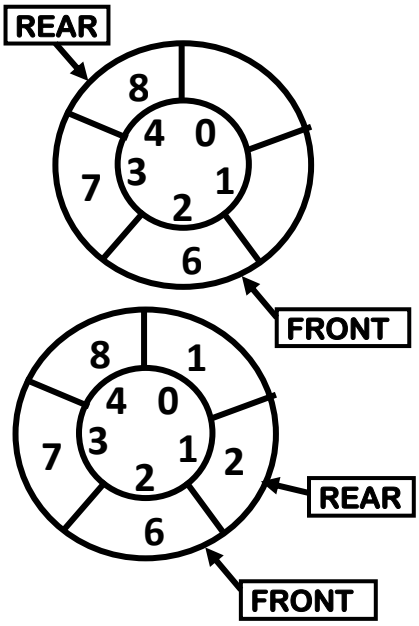
Your choice: 1
Enter the element to push: 3
QUEUE OVERFLOW.

Your choice: 3
| 6 | | 7 | | 8 | | 1 | | 2 |
Your choice: 4

TEST CASE-2: Overflow-2



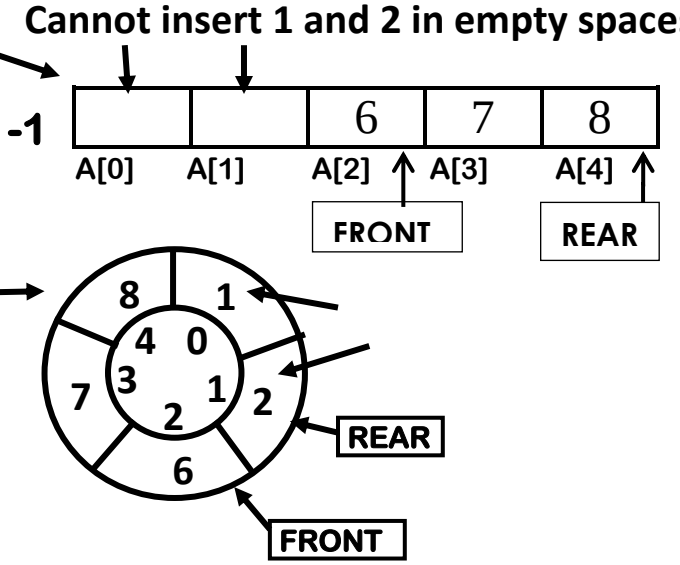
The element to be deleted is first stored in another variable(ele), and then front is incremented, ++front.



dequeue(4)
dequeue(5)

enqueue(1)
enqueue(2)

Advantage: Efficient Utilization of Memory: In **Linear Queue** when we delete any element only the front is incremented by 1, but that position is not used later. So, memory wastage increases. But in **Circular Queue** memory is utilized, if we delete any element, that position is used later on, because it is circular.



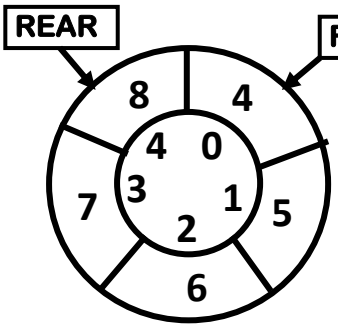
TEST CASE-3: Underflow-1

```
Your choice: 2
Dequeued 4.
Your choice: 2
Dequeued 5.
Your choice: 2
Dequeued 6.
Your choice: 2
Dequeued 7.
Your choice: 2
Dequeued 8.
Your choice: 2
UNDERFLOW CONDITION.
Your choice: 4
```

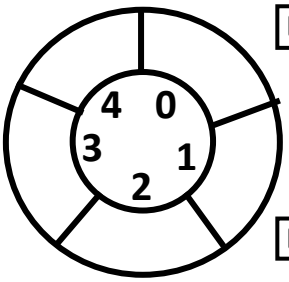
TEST CASE-4: Underflow-2

```
Your choice: 3
| 6 |   | 7 |   | 8 |   | 1 |   | 2 |
Your choice: 2
Dequeued 6.
Your choice: 2
Dequeued 7.
Your choice: 2
Dequeued 7.
Your choice: 2
Dequeued 1.
Your choice: 2
Dequeued 2.
Your choice: 2
UNDERFLOW CONDITION.
Your choice: 4
```

TEST CASE-3: Underflow-1

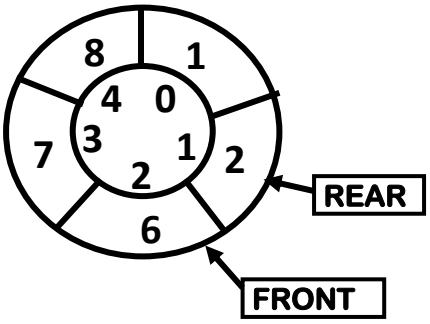


Initial state of Queue

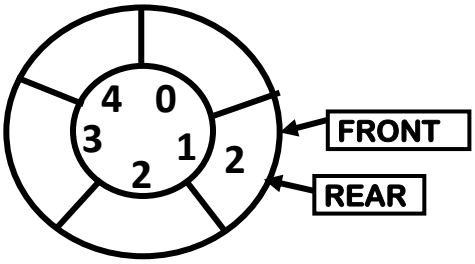


dequeue(4)
dequeue(5)
dequeue(6)
dequeue(7)
dequeue(8)

TEST CASE-4: Underflow-2

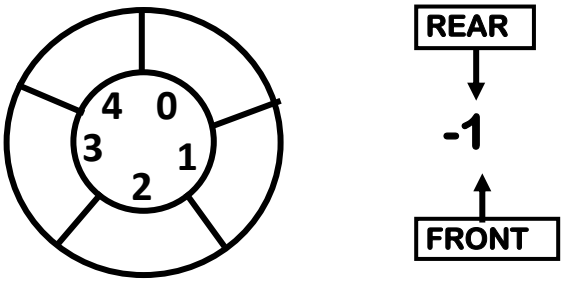


dequeue(6)
dequeue(7)
dequeue(8)
dequeue(1)



dequeue(2)

Now element '2' is stored in another variable(ele) and both the front and rear are reset to '-1'.



PRACTICAL NO:

AIM: Implement Singly Linked List using C.

The following program performs the following operations on Singly Linked List:

1. Insert an element from Beginning
2. Insert an element from End
3. Insert an element in any Intermediate Position
4. Delete an element from Beginning
5. Delete an element from End
6. Delete an element from any Intermediate Position
7. count() : It counts the number of elements in the linked list
8. display(): It displays the current elements in the linked list

PROGRAM

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

struct node
{
  int data;
  struct node *next;
};
struct node *head=NULL;

/*------Insert from Beginning------*/
void insert_beg(int ele)
{
 struct node *temp=(struct
node*)malloc(sizeof(struct node));
 temp->data=ele;
 temp->next=head;
 head=temp;
}

/*------Insert from End------*/
void insert_end(int ele)
{
  struct node *temp1=head;
  struct node *temp=(struct
node*)malloc(sizeof(struct node));
  temp->data = ele;
  temp->next = NULL;

  if(head==NULL)
    insert_beg(ele);
  else
  {
    while(temp1->next!=NULL)
```

```
   {
     temp1=temp1->next;
   }
   temp1->next=temp;
  }
}

/*------Delete from Beginning------*/
void delete_beg()
{
  struct node *temp=head;
  if(head==NULL)
    printf("Cannot delete element.
Underflow\n");
  else
  {
    head = temp->next;
    free(temp);
  }
}


/*------------Delete from End----------*/
void delete_end()
{
  struct node *temp,*temp1;
  temp=temp1=head;
  if(head==NULL)
    printf("Underflow Condition.\n");
  else if(head->next==NULL)
  {
   temp=head;
   free(temp);
   head=NULL;
  }
  else
  {
    while(temp->next!=NULL)
     {
       temp1=temp;
       temp = temp->next;
     }
    temp1->next=NULL;
    free(temp);
  }
}

/*-------Count Function----------*/
int count(void)
{
 struct node *temp=head;
 int count=1;
 while(temp->next!=NULL)
 {
  temp=temp->next;
  count++;
 }
 return (count);
}

/*-------Insert Randomly----------*/
void insert_any(int ele, int pstn)
{
  int k,i;
  struct node *temp=(struct
node*)malloc(sizeof(struct node));
  temp->data=ele;
  temp->next=NULL;

  k=count();
  if(pstn<=k+1)
  {
   if(pstn==1)
     insert_beg(ele);
   else if(pstn==k+1)
     insert_end(ele);
   else
   {
```

```
    struct node *temp1=head;
    for(i=0;i<pstn-2;i++)
    {
     temp1=temp1->next;
    }
    temp->next=temp1->next;
    temp1->next=temp;
   }
  }
  else
  {
   printf("\nThe last position of the element is %d. Hence, cannot insert %d at %d position.", k,ele,pstn);
  }
}

/*-------Delete Randomly----------*/
void delete_any(int pstn)
{
  int k,i;
  struct node *temp=head;
  struct node *temp1=head;
  k=count();

  if(pstn<=k)
  {
   if(pstn==1)
     delete_beg();
   else if(pstn==k)
      delete_end();
   else
   {
    for(i=0;i<pstn-2;i++)
     {
      temp=temp->next;
     }
    temp1=temp->next;
    temp->next=temp1->next;
     printf("\nDeleted [ %d | %d ] node.\n", temp1->data, temp1->next);
     temp1->next=NULL;

     free(temp1);
   }
  }
  else
  {
   printf("\nTotal elements: %d. Hence there is not element at %d position.\n", k, pstn);
  }
}

/*-------Display Function----------*/
void display()
{
 struct node *temp=head;
 printf("\nHead(%d)", head);
 while(temp!=NULL)
 {
  printf("-->[%d(%d) | %d]", temp->data,temp,temp->next);
  temp=temp->next;
 }
 printf("\n");
}
```

```
/*-------MAIN FUNCTION----------*/
int ele,pstn;
void main()
{
  int ch;
  clrscr();
  do
  {
  printf("\t=====MENU=======\n");
  printf("\t1] Insert from
Beginning\n");
  printf("\t2] Delete from
Beginning\n");
  printf("\t3] Insert from End\n");
  printf("\t4] Delete from End\n");
  printf("\t5] Insert at any position\n");
  printf("\t6] Delete from any
position\n");
  printf("\t7] Display.\n");
  printf("Enter your choice: ");
  scanf("%d",&ch);
    switch(ch)
    {
     case 1:
        printf("\nEnter the element you
want to insert: ");
        scanf("%d", &ele);
        insert_beg(ele);
        break;
     case 2:
        delete_beg();
        break;
     case 3:
        printf("\nEnter the element you
want to insert: ");
        scanf("%d", &ele);
        insert_end(ele);
        break;
     case 4:
        delete_end();
        break;
     case 5:
         printf("\nEnter the element to
insert: ");
         scanf("%d", &ele);
         printf("\nWhere do you want to
insert %d? ", ele);
         scanf("%d", &pstn);
         insert_any(ele,pstn);
         break;
     case 6:
         printf("\nEnter the position of
element you want to delete.");
         scanf("%d", &pstn);
         delete_any(pstn);
         break;
     case 7:
        display();
        break;
    }
  }
  while(ch);
  getch();
}
```

OUTPUT

Case 1: Insert from Beginning

Menu Repeats

```
=====MENU=======
1] Insert from Beginning
2] Delete from Beginning
3] Insert from End
4] Delete from End
5] Insert at any position
6] Delete from any position
7] Display.
Enter your choice: 1
Enter the element you want to insert: 3

Enter your choice: 1
Enter the element you want to insert: 5

Enter your choice: 1
Enter the element you want to insert: 7

Enter your choice: 7

Head(2526)-->[7(2526) | 2518]-->[5(2518) | 2510]-->[3(2510) | 0]
```

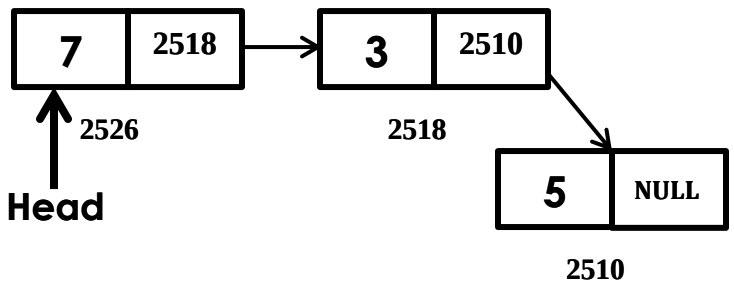
OUTPUT

Case 2: Delete from Beginning

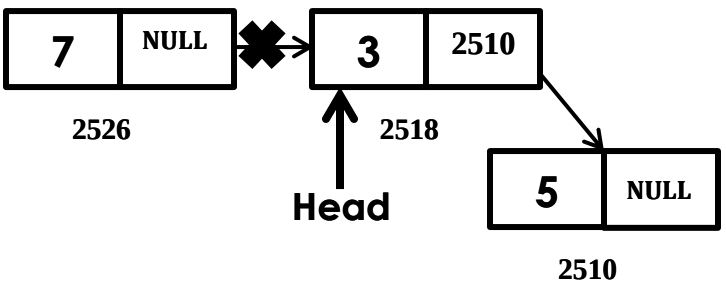
Menu Repeats

```
=====MENU=======
1] Insert from Beginning
2] Delete from Beginning
3] Insert from End
4] Delete from End
5] Insert at any position
6] Delete from any position
7] Display.
Enter your choice: 2

Enter your choice: 7

Head(2526)-->[5(2518) | 2510]-->[3(2510) | 0]
```

Diagrammatic Representation



Diagrammatic Representation



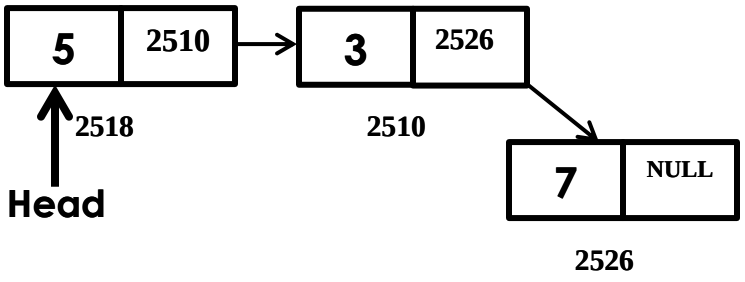
OUTPUT

Case 3: Insert from the End

Menu Repeats

```
=====MENU=======
1] Insert from Beginning
2] Delete from Beginning
3] Insert from End
4] Delete from End
5] Insert at any position
6] Delete from any position
7] Display.
Enter your choice: 3
Enter the element you want to insert: 7

Enter your choice: 7

Head(2526) -->[5(2518) | 2510]--
>[3(2510) | 2526] -->[7(2526) | 0]
```

Diagrammatic Representation



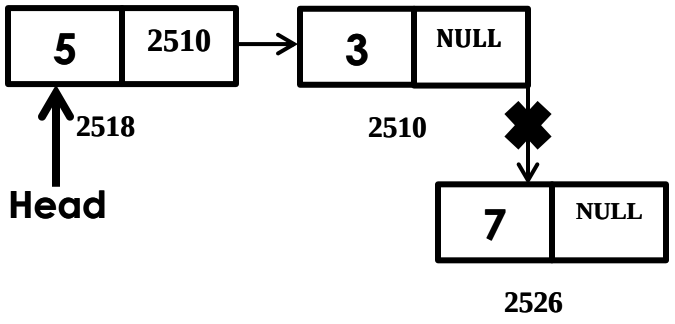
OUTPUT

Case 4: Delete from End

Menu Repeats

```
=====MENU=======
1] Insert from Beginning
2] Delete from Beginning
3] Insert from End
4] Delete from End
5] Insert at any position
6] Delete from any position
7] Display.
Enter your choice: 4

Enter your choice: 7

Head(2526) -->[5(2518) | 2510]--
>[3(2510) | 0]
```

Diagrammatic Representation



PRACTICAL NO:

AIM: Implement Circular Linked List using C.

The following program performs the following operations on Circular Linked List:

1. Insert an element from Beginning
2. Insert an element from End
3. Insert an element in any Intermediate Position
4. Delete an element from Beginning
5. Delete an element from End
6. Delete an element from any Intermediate Position
7. count() : It counts the number of elements in the linked list
8. display(): It displays the current elements in the linked list

PROGRAM

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

struct node
{
  int data;
  struct node *next;
};
struct node *front=NULL;
struct node *rear=NULL;

/*------Insert from Beginning------*/
void insert_beg(int ele)
{
 struct node *temp=(struct
node*)malloc(sizeof(struct node));
 temp->data=ele;

 if(front==NULL && rear==NULL)
 {
   front=rear=temp;
   temp->next=front;
 }
 else
 {
   temp->next=front;
   front=temp;
   rear->next=front;
 }
}
```

```
/*------Insert from End------*/
void insert_end(int ele)
{
 if(front==NULL && rear==NULL)
 {
  insert_beg(ele);
 }
 else
 {
 struct node *temp=(struct
node*)malloc(sizeof(struct node));
 temp->data=ele;
 rear->next=temp;
 rear=temp;
 rear->next=front;
 }
}

/*------Delete from Beginning------*/
void delete_beg()
{
 struct node *temp=front;
 if(temp==NULL)
   printf("Cannot delete element.
Underflow\n");
 else if(front==rear)
 {
    front=rear=NULL;
    free(temp);
 }
 else
 {
   front=temp->next;
   rear->next=front;
   temp->next=NULL;
   free(temp);
 }
}
```

```
/*------------Delete from End----------*/
void delete_end()
{
 struct node *temp;
 temp=front;
 if(temp==NULL)
   printf("Underflow Condition.\n");
 else if(front==rear)
 {
    front=rear=NULL;
    free(temp);
 }
 else
 {
  while(temp->next!=rear)
    {
      temp=temp->next;
    }
   rear=temp;
   temp=temp->next;
   rear->next=front;
   temp->next=NULL;
   free(temp);
 }
}
/*-------Count Function----------*/
int count(void)
{
 struct node *temp=front;
 int count=1;
 while(temp->next!=front)
 {
  temp=temp->next;
  count++;
 }
 return(count);
}
```

```
/*-------Insert Randomly----------*/
void insert_any(int ele, int pstn)
{
  int k,i;
  k=count();
  if(pstn<=k+1)
  {
   if(pstn==1)
     insert_beg(ele);
   else if(pstn==k+1)
     insert_end(ele);
   else
   {
    struct node *temp1=front;
    struct node *temp=(struct
node*)malloc(sizeof(struct node));
    temp->data=ele;
    temp->next=NULL;

    for(i=0;i<pstn-2;i++)
    {
     temp1=temp1->next;
    }
    temp->next=temp1->next;
    temp1->next=temp;
   }
  }
  else
  {
    printf("\nThe last position of the
element is %d. Hence, cannot insert
%d at %d position.", k,ele,pstn);
  }
}
```

```
/*-------Delete Randomly----------*/
void delete_any(int pstn)
{
  int k,i;
  struct node *temp=front;
  struct node *temp1=front;
  k=count();

  if(pstn<=k)
  {
   if(pstn==1)
     delete_beg();
   else if(pstn==k)
      delete_end();
   else
   {
    for(i=0;i<pstn-2;i++)
     {
      temp=temp->next;
     }
     temp1=temp->next;
     temp->next=temp1->next;

     printf("\nDeleted [ %d | %d ]
node.\n", temp1->data, temp1->next);
     temp1->next=NULL;

     free(temp1);
   }
  }
  else
  {
   printf("\nTotal elements: %d. Hence
there is not element at %d position.\n",
k, pstn);
  }
}
```

```
/*-------Display Function----------*/
void display()
{
 struct node *temp=front;
 printf("\nFront(%d)", front);
 do
 {
   printf("-->[%d(%d) | %d]", temp->data,temp,temp->next);
   temp=temp->next;
 }
 while(temp!=front);
 printf("<--Rear(%d)", rear);

 printf("\n");
}

/*-------MAIN FUNCTION----------*/
int ele,pstn;
void main()
{
  int ch;
  clrscr();
  do
  {
  printf("\t=====MENU=======\n");
  printf("\t1] Insert from Beginning\n");
  printf("\t2] Delete from Beginning\n");
  printf("\t3] Insert from End\n");
  printf("\t4] Delete from End\n");
  printf("\t5] Insert at any position\n");
  printf("\t6] Delete from any position\n");
  printf("\t7] Display.\n");
  printf("Enter your choice: ");
  scanf("%d",&ch);
    switch(ch)    {
      case 1:
         printf("\nEnter the element you want to insert: ");
         scanf("%d", &ele);
         insert_beg(ele);
         break;
      case 2:
         delete_beg();
         break;
      case 3:
         printf("\nEnter the element you want to insert: ");
         scanf("%d", &ele);
         insert_end(ele);
         break;
      case 4:
         delete_end();
         break;
      case 5:
          printf("\nEnter the element to insert: ");
          scanf("%d", &ele);
          printf("\nWhere do you want to insert %d? ", ele);
          scanf("%d", &pstn);
          insert_any(ele,pstn);    break;
      case 6:
          printf("\nEnter the position of element you want to delete.");
          scanf("%d", &pstn);
          delete_any(pstn);
          break;
      case 7:
         display();   break;
     }
   }
  while(ch);
  getch();
}
```

OUTPUT

=====MENU=======
1] Insert from Beginning
2] Delete from Beginning
3] Insert from End
4] Delete from End
5] Insert at any position
6] Delete from any position
7] Display.

Menu Repeats

Case 1: Insert from Beginning

Enter your choice: 1
Enter the element you want to insert: 3

Enter your choice: 1
Enter the element you want to insert: 4

Enter your choice: 1
Enter the element you want to insert: 5
Enter your choice: 7
Front(2354)-->[5(2354) | 2346]-->[4(2346) | 2338]-->[3(2338) | 2354]<--Rear(2338)

Case 2: Delete from End

Enter your choice: 2
Enter your choice: 7
Front(2346)-->[4(2346) | 2338]-->[3(2338) | 2346]<--Rear(2338)

Case 3: Insert from End

Enter your choice: 3
Enter the element you want to insert: 5

Enter your choice: 7
Front(2346)-->[4(2346) | 2338]-->[3(2338) | 2354]-->[5(2354) | 2346]<--Rear(2354)

Case 5: Intermediate Insertion

Enter your choice: 5
Enter the element to insert: 7
Where do you want to insert 7? 3

Enter your choice: 7
Front(2354)-->[4(2354) | 2346]-->[3(2346) | 2362]-->[7(2362) | 2338]-->[5(2338) | 2354]<--Rear(2338)

Case 6: Intermediate Deletion

Enter your choice: 6
Enter the position of element you want to delete:2
Deleted [3 | 2362] node.

Enter your choice: 7
Front(2354)-->[4(2354) | 2362]-->[7(2362) | 2338]-->[5(2338) | 2354]<--Rear(2338)

Enter your choice: 0

PRACTICAL NO:

AIM: Implement Doubly Linked List using C.

The following program performs the following operations on Doubly Linked List:

1. Insert an element from Beginning
2. Insert an element from End
3. Insert an element in any Intermediate Position
4. Delete an element from Beginning
5. Delete an element from End
6. Delete an element from any Intermediate Position
7. count() : It counts the number of elements in the linked list
8. display(): It displays the current elements in the linked list

PROGRAM

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

struct node
{
  int data;
  struct node *next;
  struct node *prev;
};
struct node *head=NULL;

/*-------newnode function-------*/
struct node* newnode(int ele)
{
  struct node *temp=(struct
node*)malloc(sizeof(struct node));
  temp->data=ele;
  temp->next=NULL;
  head->prev=NULL;
  return(temp);
}

/*------Insert from Beginning------*/
void insert_beg(int ele)
{
 struct node *temp=newnode(ele);
 temp->next=head;
 head->prev=temp;
 head=temp;
}
```

```
/*------Insert from End------*/
void insert_end(int ele)
{
 if(head==NULL)
  insert_beg(ele);
 else
 {
  struct node *temp1=head;
  struct node *temp=newnode(ele);
  while(temp1->next!=NULL)
  {
   temp1=temp1->next;
  }
  temp1->next=temp;
  temp->prev=temp1;
 }
}

/*------Delete from Beginning------*/
void delete_beg()
{
 struct node *temp=head;
 if(head==NULL)
  printf("Cannot delete element.
Underflow\n");
 else if(head->next==NULL)
 {
  temp=head;
  head=NULL;
  free(temp);
 }
 else
 {
  head = temp->next;
  free(temp);
  head->prev=NULL;
 }
}
```

```
/*------------Delete from End----------*/
void delete_end()
{
 if(head==NULL)
  printf("Underflow Condition.\n");
 else if(head->next==NULL)
 {
  delete_beg();
 }
 else
 {
  struct node *temp,*temp1;
  temp=temp1=head;
  while(temp->next!=NULL)
       temp = temp->next;
  temp1=temp->prev;
  temp1->next=NULL;
  free(temp);
 }
}

/*-------Count Function----------*/
int count(void)
{
 struct node *temp=head;
 int count=1;
 if(head==NULL)
  {
   count=0;
   return(count);
  }
 while(temp->next!=NULL)
 {
  temp=temp->next;
  count++;
 }
 return (count);
}
```

```
/*-------Insert Randomly----------*/
void insert_any(int ele, int pstn)
{
  int k,i;
  k=count();
  if(pstn<=k+1)
  {
   if(pstn==1)
     insert_beg(ele);
   else if(pstn==k+1)
     insert_end(ele);
   else
   {
    struct node *temp=newnode(ele);
    struct node *temp1=head;
    for(i=0;i<pstn-2;i++)
    {
     temp1=temp1->next;
    }
    temp->next=temp1->next;
    temp1->next=temp;
    temp->prev=temp1;
    temp1=temp->next;
    temp1->prev=temp;
   }
  }
  else
  {
    printf("\nThe position of the last
element is %d. Hence, cannot insert
%d at %d position.", k,ele,pstn);
  }
}
```

```
/*-------Delete Randomly----------*/
void delete_any(int pstn)
{
  int k,i;
  k=count();

  if(pstn<=k)
  {
   if(pstn==1)
     delete_beg();
   else if(pstn==k)
      delete_end();
   else
   {
     struct node *temp=head;
     struct node *temp1=head;
    for(i=0;i<pstn-2;i++)
     {
      temp=temp->next;
     }
     temp1=temp->next;
     temp->next=temp1->next;
     temp=temp1->next;
     temp->prev=temp1->prev;
     printf("\nDeleted [ %d | %d | %d ]
node.\n", temp1->prev, temp1->data,
temp1->next);
     free(temp1);
   }
  }
  else
  {
   printf("\nTotal elements: %d. Hence
there is no element at %d position.\n",
k, pstn);
  }
}
```

```
/*-------Display Function----------*/
void display()
{
 struct node *temp=head;
 printf("\nHead(%d)", head);
 while(temp!=NULL)
 {
   printf("<-->[%d|%d(%d)|%d]",
temp->prev,temp->data,temp,temp-
>next);
   temp=temp->next;
 }
 printf("\n");
}

/*-------MAIN FUNCTION----------*/
int ele,pstn;
void main()
{
  int ch;
  clrscr();
  do
  {
  printf("\t=====MENU=======\n");
  printf("\t1] Insert from
Beginning\n");
  printf("\t2] Delete from
Beginning\n");
  printf("\t3] Insert from End\n");
  printf("\t4] Delete from End\n");
  printf("\t5] Insert at any position\n");
  printf("\t6] Delete from any
position\n");
  printf("\t7] Display.\n");
  printf("Enter your choice: ");
  scanf("%d",&ch);
    switch(ch)
    {
     case 1:
        printf("\nEnter the element you
want to insert: ");
        scanf("%d", &ele);
        insert_beg(ele);
        break;
     case 2:
        delete_beg();      break;
     case 3:
        printf("\nEnter the element you
want to insert: ");
        scanf("%d", &ele);
        insert_end(ele);
        break;
     case 4:
        delete_end();
        break;
     case 5:
         printf("\nEnter the element to
insert: ");
         scanf("%d", &ele);
         printf("\nWhere do you want to
insert %d? ", ele);
         scanf("%d", &pstn);
         insert_any(ele,pstn);
         break;
     case 6:
         printf("\nEnter the position of
element you want to delete.");
         scanf("%d", &pstn);
         delete_any(pstn);
         break;
     case 7:
        display();
        break;
    }
  }
  while(ch);
  getch();
}
```

OUTPUT

=====MENU=======
1] Insert from Beginning
2] Delete from Beginning
3] Insert from End
4] Delete from End
5] Insert at any position
6] Delete from any position
7] Display.

Menu Repeats

Enter your choice: 1
Enter the element you want to insert: 1

Enter your choice: 1
Enter the element you want to insert: 3

Enter your choice: 1
Enter the element you want to insert: 2

Enter your choice: 7
Head(2350)<-->[0|2(2350)|2340]<-->[2350|3(2340)|2330]<-->[2340|1(2330)|0]

Case 2: Delete from Beginning

Enter your choice: 2

Enter your choice: 7
Head(2340)<-->[0|3(2340)|2330]<-->[2340|1(2330)|0]

Case 3: Insert from the End

Enter your choice: 3
Enter the element you want to insert: 4

Enter your choice: 7
Head(2340)<-->[0|3(2340)|2330]<-->[2340|1(2330)|2350]<-->[2330|4(2350)|0]

Case 4: Delete from the End

Enter your choice: 4

Enter your choice: 7
Head(2340)<-->[0|3(2340)|2330]<-->[2340|1(2330)|0]

Case 5: Intermediate Insertion

Enter your choice: 5
Enter the element to insert: 2
Where do you want to insert 2? 2

Enter your choice: 7
Head(2340)<-->[0|3(2340)|2350]<-->[2340|2(2350)|2330]<-->[2350|1(2330)|0]

Case 6: Intermediate Deletion

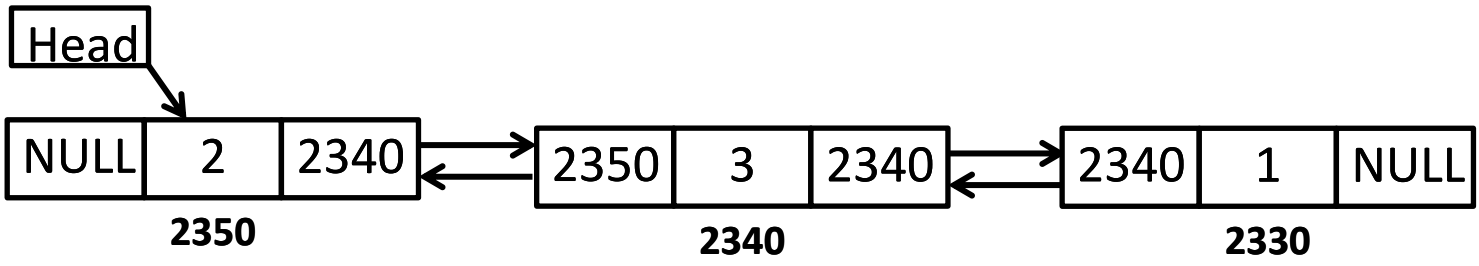
Enter your choice: 6
Enter the position of element you want to delete.2
Deleted [2340 | 2 | 2330] node.

Enter your choice: 7
Head(2340)<-->[0|3(2340)|2330]<-->[2340|1(2330)|0]

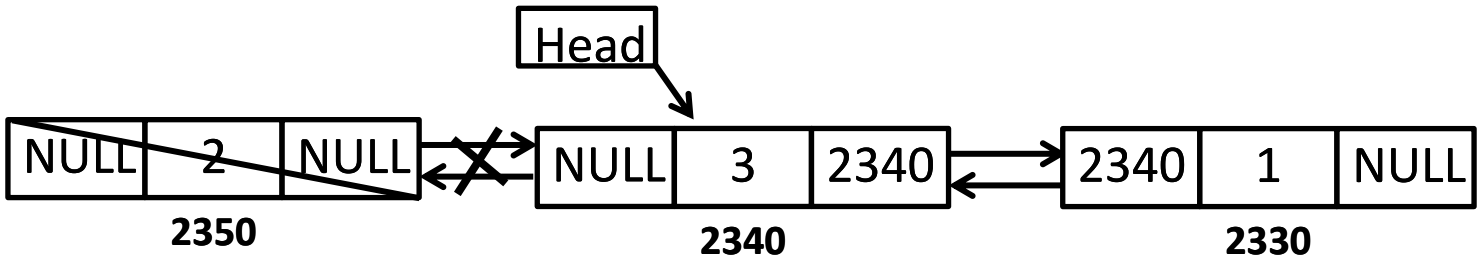
Enter your choice: 0

Graphical Representation

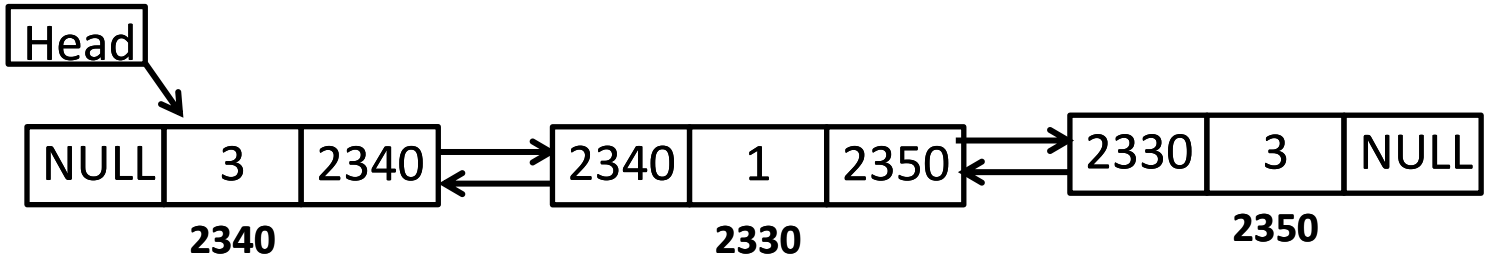
Case 1: Insert from Beginning



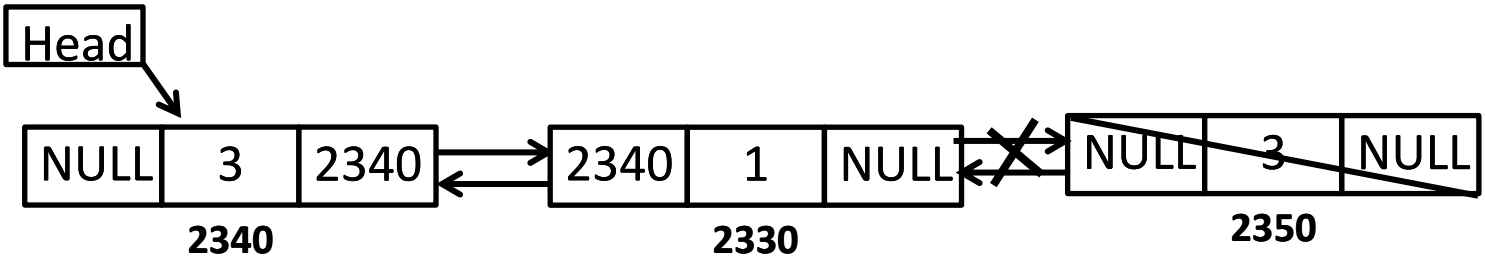
Case 2: Delete from Beginning



Case 3: Insert from the End



Case 4: Delete from the End



PRACTICAL NO:

AIM: Implement Addition of two polynomials using Linked List.

The following program performs the addition operation on two polynomials using Doubly Linked List.

It includes the following functions:

1. newnode( ): Generates new node dynamically.
2. Insert_p1( ): Creates node for every term of polynomial – 1 and links it to p1 head node.
3. Insert_p2( ): Creates node for every term of polynomial – 2 and links it to p1 head node.
4. Insert_sum( ): The resultant summation of two polynomials is stored in this linked list.
5. Addition( ): Has the actual logic, to add given two polynomials.
6. Initialize_poly( ): It reads the values of the two polynomials,
7. Display( ): Displays the required resultant sum linked list.

PROGRAM

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

void display(void);

struct node
{
  int coef, expo;
  struct node *next;
};
struct node *p1=NULL;
struct node *p2=NULL;
struct node *sum=NULL;

/*------Newnode Function------*/
struct node* newnode(int num1, int
num2)
{
  struct node *temp=(struct
node*)malloc(sizeof(struct node));
  temp->coef=num1;
  temp->expo=num2;
  temp->next=NULL;
  return(temp);
}
```

/*------Polynomial – 1 Function-----*/

```
void insert_p1(int num1, int num2)
{
 struct node
*temp=newnode(num1,num2);
 if(p1==NULL)
   {
     p1=temp;
   }
 else
 {
  struct node *temp1=p1;
  while(temp1->next!=NULL)
  {
    temp1=temp1->next;
  }
  temp1->next=temp;
 }
}
```

/*------Polynomial – 2 Function-----*/

```
void insert_p2(int num1, int num2)
{
  struct node
*temp=newnode(num1,num2);
 if(p2==NULL)
   {
     p2=temp;
   }
 else
 {
  struct node *temp1=p2;
  while(temp1->next!=NULL)
  {
    temp1=temp1->next;
  }
  temp1->next=temp;
 }
}
```

/*--------Sum Linked List---------*/

```
void insert_sum(int num1, int num2)
{
 struct node
*temp=newnode(num1,num2);
 if(sum==NULL)
   {
     sum=temp;
   }
 else
 {
  struct node *temp1=sum;

  while(temp1->next!=NULL)
  {
    temp1=temp1->next;
  }
  temp1->next=temp;
 }
}
```

/*-------Display Function----------*/

```
void display()
{
 struct node *temp=sum;
 printf("\nHead(%d)", sum);
 while(temp!=NULL)
 {
  printf("-->[%d|%d(%d)|%d]", temp-
>coef,temp->expo,temp,temp->next);
  temp=temp->next;
 }
 printf("\n");
 p1=p2=sum=NULL;
 printf("\nThe pointers are now reset to
NULL. New values can be entered.
\n");
 }
```

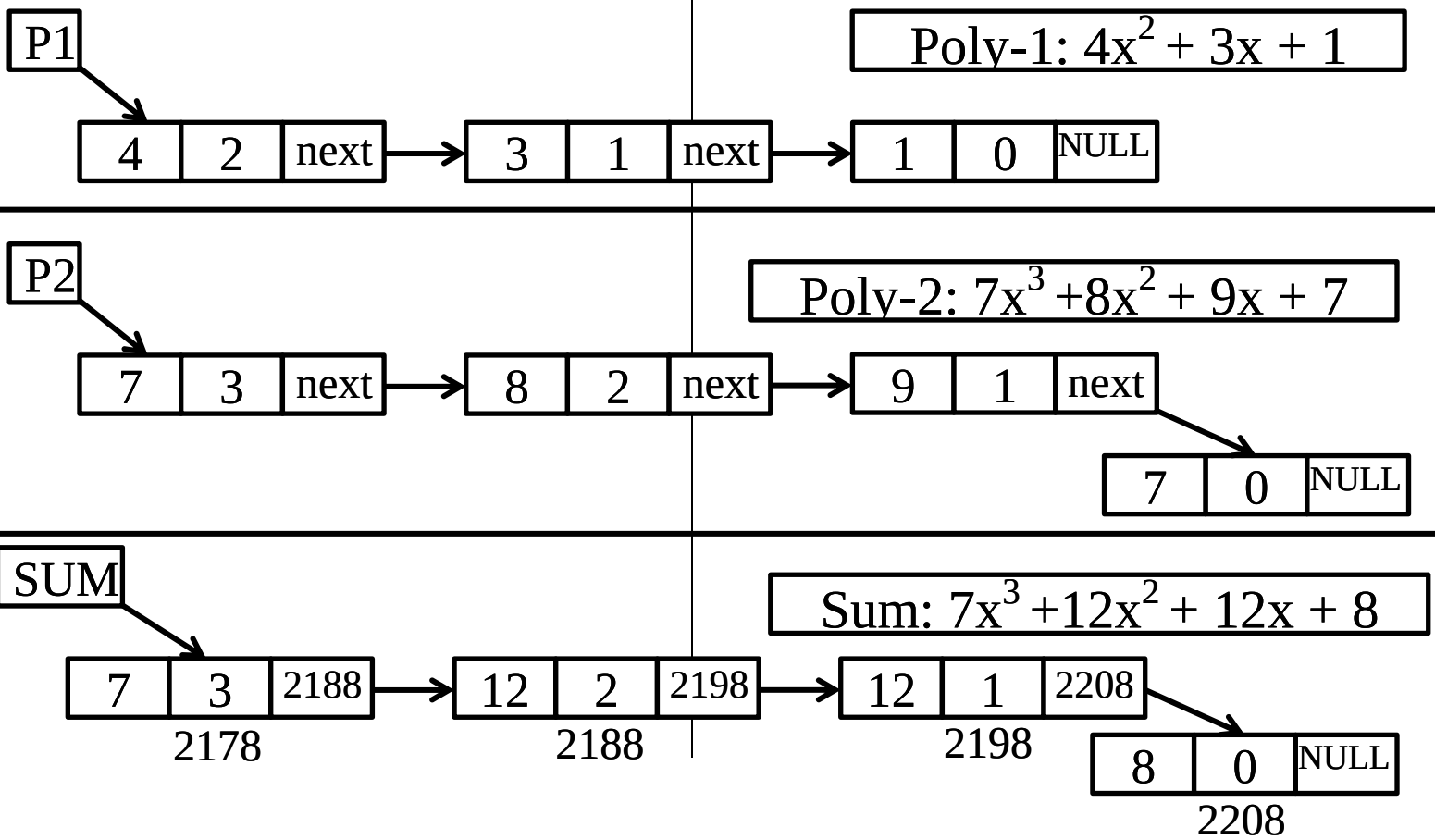
```
/*---------Addition Operation---------*/
void addition()
{
 int k;
 while(p1!=NULL && p2!=NULL)
 {
   if(p1->expo==p2->expo)
   {
      k=p1->coef+p2->coef;
      insert_sum(k,p1->expo);
      p1=p1->next;
      p2=p2->next;
      addition();
   }

    if(p1->expo<p2->expo)
   {
     insert_sum(p2->coef,p2->expo);
     p2=p2->next;
     addition();
   }

   if(p1->expo>p2->expo)
   {
     insert_sum(p1->coef,p1->expo);
     p1=p1->next;
     addition();
   }
   if(p1==NULL)
   {
    while(p2!=NULL)
     {
       insert_sum(p2->coef,p2->expo);
       p2=p2->next;
     }
   addition();
   }

   if(p2==NULL)
     {
      while(p1!=NULL)
      {
        insert_sum(p1->coef,p1->expo);
        p1=p1->next;
      }
      addition();
     }
  }
}

/*-------Initializing polynomials------*/
void initialize_poly()
{
  int i,n;
  printf("\nEnter the no. of terms of
Poly-1: ");
   scanf("%d", &n);
   for(i=0;i<n;i++)
   {
      printf("Enter Coefficient and
Exponent: ");
      scanf("%d %d", &num1, &num2);
      insert_p1(num1,num2);
   }
   printf("\nEnter the no. of terms of
Poly-2: ");
   scanf("%d", &n);
   for(i=0;i<n;i++)
   {
      printf("Enter Coefficient and
Exponent: ");
      scanf("%d %d", &num1, &num2);
      insert_p2(num1,num2);
   }
}
```

```
/*-------MAIN FUNCTION----------*/
int num1,num2;
void main()
{
  int ch;
  clrscr();
   do
   {
    initialize_poly();
    addition();
    display();
    printf("Continue? (1/0): ");
    scanf("%d", &ch);
   }
  while(ch);
  getch();
}
```

OUTPUT

```
Enter the no. of terms of Poly-1: 3
Enter Coefficient and Exponent: 4 2
Enter Coefficient and Exponent: 3 1
Enter Coefficient and Exponent: 1 0

Enter the no. of terms of Poly-2: 4
Enter Coefficient and Exponent: 7 3
Enter Coefficient and Exponent: 8 2
Enter Coefficient and Exponent: 9 1
Enter Coefficient and Exponent: 7 0

Head(2178)-->[7|3(2178)|2188]--
>[12|2(2188)|2198]--
>[12|1(2198)|2208]-->[8|0(220
8)|0]

The pointers are now reset to NULL.
New values can be entered.
Continue? (1/0): 0
```

Graphical Representation



PRACTICAL NO:

AIM: Implement Pre-order, In-order and Post-order on a Tree using C Program.

PROGRAM

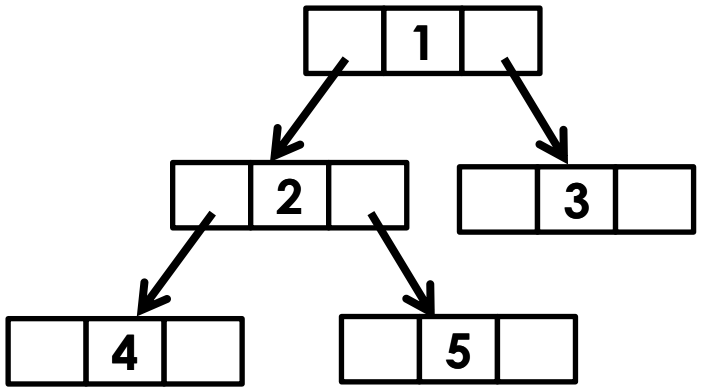
```
#include<stdio.h>
#include<conio.h>

struct node
{
  struct node *left;
  int data;
  struct node *right;
};
struct node *root;

/*------Newnode Function------*/
struct node* newnode(int ele)
{
 struct node *temp=(struct
node*)malloc(sizeof(struct node));
 temp->left=NULL;
 temp->right=NULL;
 temp->data=ele;
 return (temp);
}

/*------Preorder Function-----*/
void Preorder(struct node *Node)
{
 if(Node==NULL)
   return;
 printf(" %d ", Node->data);
 Preorder(Node->left);
 Preorder(Node->right);
}

/*------Inorder Function-----*/
void Inorder(struct node *Node)
{
 if(Node==NULL)
   return;
 Inorder(Node->left);
 printf(" %d ", Node->data);
 Inorder(Node->right);
}

/*--------PostOrder Function---------*/
void Postorder(struct node *Node)
{
 if(Node==NULL)
    return;
 Postorder(Node->left);
 Postorder(Node->right);
 printf(" %d ", Node->data);
}
```

```
/*-------MAIN FUNCTION----------*/
void main()
{

  struct node *root=newnode(1);
    clrscr();
  root->left=newnode(2);
  root->right=newnode(3);
  root->left->left=newnode(4);
  root->left->right=newnode(5);

  printf("\nPreorder Traversal of
Binary Tree is: ");
  Preorder(root);
  printf("\nInorder Traversal of
Binary Tree: ");
  Inorder(root);
  printf("\nPostorder Traversal of
Binary Tree: ");
  Postorder(root);

  getch();
}
```

OUTPUT

```
Preorder Traversal of Binary Tree
is:  1  2  4  5  3
Inorder Traversal of Binary Tree:
4  2  5  1  3
Postorder Traversal of Binary
Tree:  4  5  2  3  1
```

Graphical Representation



PRACTICAL NO:

AIM: Implement Adjacency List Representation of Graph using a C Program.

PROGRAM

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

struct node
{
 int data;
 struct node *next;
};

struct graph
{
 int v;
 int e;
 struct node *list;
};

struct graph *g; /*This pointer points to the graph structure*/
```

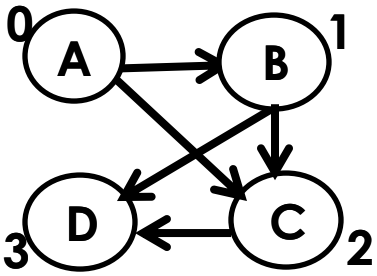
```
/*------Adj. List Logic Function------*/
void create_graph()
{
  int src,dest,i;
  struct node *temp, *temp1, *t,*t1;
g=(struct graph*)malloc(sizeof(struct graph));
  printf("\nEnter the no. of vertices and edges: ");
  scanf("%d %d", &g->v,&g->e);
  g->list=NULL;
  g->list=(struct node*)malloc(g->v * (sizeof(struct node)));

  for(i=0;i<g->v;i++)
  {
    t=g->list+i;
    t->data = i;
    t->next=NULL;
  }

  for(i=0;i<g->e;i++)
  {
   printf("\nConnections: ");
   scanf("%d %d", &src,&dest);
   temp=(struct node*)malloc(sizeof(struct node));
     temp->data=dest;
     temp->next=NULL;
     t1=g->list+src;

   if(t1->next==NULL)
        t1->next=temp;
```

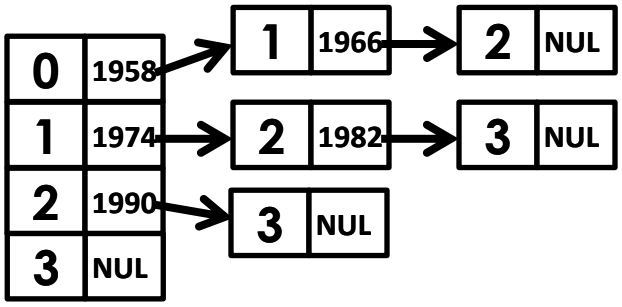
```
  else
   {
    temp1=t1;
    while(temp1->next!=NULL)
       temp1=temp1->next;
    temp1->next=temp;
   }
  }

  for(i=0;i<g->v;i++)
  {
   struct node *temp2;
   temp2=g->list+i;
   printf("\n| %d(%d) | %d |",temp2-
>data,temp2,temp2->next);
   while(temp2->next!=NULL)
   {
    temp2=temp2->next;
    printf("-->| %d(%d) | %d |",temp2-
>data,temp2,temp2->next);
   }
  }

}

/*-------MAIN FUNCTION----------*/
void main()
{
 clrscr();
 create_graph();
 getch();
}
```

OUTPUT

```
Enter the no. of vertices and
edges: 4 5
Connections: 0 1
Connections: 0 2
Connections: 1 2
Connections: 1 3
Connections: 2 3

| 0(1938) | 1958 |-->| 1(1958) |
1966 |-->| 2(1966) | 0 |
| 1(1942) | 1974 |-->| 2(1974) |
1982 |-->| 3(1982) | 0 |
| 2(1946) | 1990 |-->| 3(1990) | 0 |
| 3(1950) | 0 |
```

Graphical Representation





PRACTICAL NO:

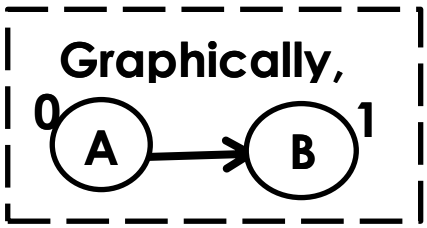
AIM: Implement Adjacency Matrix Representation of Graph using a C Program.

PROGRAM

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

struct graph
{
 int nodes;
 int edges;
 int **mat;
};

/*------Adj. List Logic Function------*/

void create_graph()
{
  int i,j,k,x,y;
  struct graph *g=(struct
graph*)malloc(sizeof(struct graph));
  printf("Enter the number of nodes
and edges: ");
  scanf("%d %d",&g->nodes,&g-
>edges);
  g->mat=(int**)malloc(sizeof(int)*(g-
>nodes * g->nodes));
  for(i=0;i<g->nodes;i++)
  {
   for(j=0;j<g->nodes;j++)
   {
     g->mat[i][j]=0;
   }
  }
  printf("Enter Connections: ");
  for(i=0;i<g->edges;i++)
  {
   scanf("%d %d", &x, &y);
   g->mat[x][y]=1;
   //g->mat[y][x]=1;
  }
  for(i=0;i<g->nodes;i++)
  {
   for(j=0;j<g->nodes;j++)
   {
     printf("| %d |\t", g->mat[i][j]);
   }
   printf("\n");
  }
}

/*-------MAIN FUNCTION----------*/
void main()
{
 clrscr();
 create_graph();
 getch();
}
```

OUTPUT

```
Enter the number of nodes and
edges: 2 1
Enter Connections: 0 1
| 0 |   | 1 |
| 0 |   | 0 |
```



REFERENCES

- Class notes, by Bhagyashri Kapre Mam.
- 'Data Structures and Algorithms Made Easy' by Narasimha Kurumanchi.
- http://www.geeksforgeeks.org
- https://stackoverflow.com/
- http://www.sanfoundry.com/
- https://github.com/
- http://www.c4learn.com