LAB TASKS

***Recursion and Array*.**

1. Find the n^{th} item of the Fibonacci series (starting with 0, 1).
2. Find the factorial of an entered number.
3. Multiply two numbers using additive method.
4. Read ten numbers from user and display largest and second largest.

***Stack*.**

1. Implementation:
    a. TOS varying.
    b. TOS fixed.
2. Application:
    a. Conversion from Infix to Postfix.
    b. Evaluation of Postfix Expression.

Queue.

1. Linear Queue:
    a. Head and tail varying.
    b. Head fixed and tail varying.
2. Circular Queue.

Recursion and Tower of Hanoi.

1. WAP for:
    a. Sum of two integers (a + b).
    b. Exponential value (x^y).
    c. Sum of n natural numbers.
    d. Fibonacci series.
2. TOH.

Static Linked List.

1. Implementation of Static Linear Linked List.
2. Static Linear Linked List as Queue.

Dynamic Singly Linked List.

1. Linear Dynamic Singly Linked List.
2. Circular Dynamic Singly Linked List.

Dynamic Doubly Linked List.

1. Linear Dynamic Doubly Linked List.
2. Circular Dynamic Doubly Linked List.

Binary Search Tree (BST).

1. Basic Operations (Insert, Search) and Traversal (Pre-Order, In-Order, Post-Order) on BST.

Sorting Algorithms.

1. Bubble Sort.
2. Merge Sort.

Recursion and Array

1. Find the n^{th} item of the Fibonacci series (starting with 0, 1).

```
#include<stdio.h>
#include<conio.h>
int fib(int);

void main()
{
	int n;
	clrscr();
	printf("Enter how many number :- ");
	scanf("%d",&n);
	if(n==0)
	{
		printf("There are no items in the series.");
	}
	else
	{
		printf("The nth item of series is :- %d",fib(n));
	}
	getch();
}


int fib(int a)
{
	if (a<2)
	{
		return a;
	}
	else
	{
		return fib(a-1)+fib(a-2);
	}
}
```

2. Find the factorial of an entered number.

```
#include<stdio.h>
#include<conio.h>
int fact(int);
```

```
void main()
{
	int no;
	clrscr();
	printf("Enter the number :- ");
	scanf("%d",&no);
	printf("Factorial = %d",fact(no));
	getch();
}

int fact(int n)
{
	if (n>1)
	{
		return n*fact(n-1);
	}
	else
	{
		return 1;
	}
}
```

3. Multiply two numbers using additive method.

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

void main()
{
	int no1,no2;
	clrscr();
	printf("Enter two number :- ");
	scanf("%d%d",&no1,&no2);
	printf("Multiplication:\n\t\t%d * %d = %d",no1,no2,mult(no1,no2));
	getch();
}

int mult(int n1, int n2)
{
	if (n2>1)
	{
```

```
		return n1+mult(n1,n2-1);
	}
	else
	{
		return n1;
	}
}
```

4. Read ten numbers from user and display largest and second largest.

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

void main()
{
	int i,a[10],g,sg;
	clrscr();
	for(i=0;i<10;i++)
	{
		printf("Enter %dth no :- ", i+1);
		scanf("%d",&a[i]);
	} g=a[0];
	sg=a[0];

	for(i=0;i<10;i++)
	{
		if(g<a[i]&&sg<a[i])
		{
			sg=g;
			g=a[i];
		}
		if(a[i]<g&&a[i]>sg)
		{
			sg=a[i];
		}
	}
	printf("Largest number = %d",g);
	printf("\nSecond Largest number = %d",sg);
	getch();
}
```

Stack

1. Implementation:

a. TOS varying.

Step 1: Start.

Step 2: Declare and Initialize necessary variables.

- **tos = -1**; top of stack.
- **MAXSIZE**; a constant for maximum size the stack can hold.
- **stack[]**; an array variable with limit of MAXSIZE.

Step 3: For **PUSH** operation;

```
If tos equals maximum size of stack
        print “Stack is FULL”
Else
        Increase tos
        Read data from user to be pushed
        Place data at top of stack
```

Step 4: For **POP** operation;

```
If tos equals its initial value
        print “Stack is EMPTY”
Else
        Pop data from top of stack
        Decrease tos
        Display the popped data
```

Step 5: For **PEEK** operation;

```
If tos equals its initial value
        print “Stack is EMPTY”
Else
        Print all i^th data of stack
        If tos equals maximum size of stack
                print “Stack is FULL"
```

Step 6: Stop.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define maxsize 3
void ins();
void del();
void dis();
int tos=-1, stack[maxsize];
```

```
void main()
{
	int ch;
	while(1)
	{
		clrscr();
		printf("\n\nStack:\n\n\t1. PUSH\n\n\t2. POP\n\n\t3. PEEK\n\n\t4. EXIT");
		printf("\n------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				ins();
				break;
			case  2:
				del();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}



void ins()
{
	int data;
	if(tos==maxsize-1)
	{
		printf("\n\tStack FULL");
	}
	else
	{
		tos++;
```

```
printf("\n\tEnter data
:- ");
scanf("%d",&data);
```

```
                stack[tos]=data;
                printf("\n\tDATA PUSHED");
        }
}

void del()
{
        int data;
        if(tos==-1)
        {
                printf("\n\tStack EMPTY");
        }
        else
        {
                data=stack[tos];
                tos--;
                printf("\n\tData = %d",data);
                printf("\n\n\tDATA POPPED");
        }
}


void dis()
{
        int i;
        if(tos==-1)
        {
                printf("\n\tStack EMPTY");
        }
        else
        {
                printf("\n");
                for(i=0;i<=tos;i++)
                {
                        printf("%d\t",stack[i]);
                }
                printf("(TOS)");
                if(tos==maxsize-1)
                {
                        printf("\n\n\tStack FULL");
                }
        }
}
```

b. TOS fixed.

Step 1: Start.

Step 2: Declare and Initialize necessary variables.

- **tos = 0**; top of stack (it is always fixed to 0).
- **bos = 0**; bottom of stack.
- **MAXSIZE**; a constant for maximum size the stack can hold.
- **stack[]**; an array variable with limit of MAXSIZE.

Step 3: For **PUSH** operation;

```
If bos equals maximum size of stack
        print "Stack is FULL"
Else
        Increase bos
        Read data from user to be pushed
        Shift all the present data to its respective upper index
        Place data at top of stack
```

Step 4: For **POP** operation;

```
If bos equals tos
        print "Stack is EMPTY"
Else
        Pop data from top of stack
        Shift all the present data to its respective bottom index
        Decrease bos
        Display the popped data
```

Step 5: For **PEEK** operation;

```
If bos equals tos
        print "Stack is EMPTY"
Else
        Print all i^th data of stack
        If bos equals  maximum size of stack
                print "Stack is FULL"
```

Step 6: Stop.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define maxsize 3
#define tos 0
void ins();
void del();
void dis();
```

```
int bos=0, stack[maxsize];
```

```
void main()
{
	int ch;
	while(1)
	{
		clrscr();
		printf("\n\nStack:\n\n\t1. PUSH\n\n\t2. POP\n\n\t3. PEEK\n\n\t4. EXIT");
		printf("\n------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				ins();
				break;
			case  2:
				del();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}



void ins()
{
	int i,data;
	if(bos==maxsize)
	{
		printf("\n\tStack FULL");
	}
	else
	{
		bos++;
```

```
printf("\n\tE
nter data :");
scanf("%d",
&data);
```

```
                for(i=bos;i>tos;i--)
                {
                        stack[i]=stack[i-1];
                }
                stack[tos]=data;
                printf("\n\tDATA PUSHED");
        }
}

void del()
{
        int i,data;
        if(bos==tos)
        {
                printf("\n\tStack EMPTY");
        }
        else
        {
                data=stack[tos];
                for(i=tos;i<bos;i++)
                {
                        stack[i]=stack[i+1];
                }
                bos--;

                printf("\n\tData = %d",data);
                printf("\n\n\tDATA POPPED");
        }
}

void dis()
{
        int i;
        if(bos==tos)
        {
                printf("\n\tStack EMPTY");
        }
        else
        {
                printf("\n");
                printf("(TOS)\t");
                for(i=0;i<bos;i++)
                {
```

```
                printf("%d\t",stack[i]);
            }
            if(bos==maxsize)
            {
                printf("\n\n\tStack FULL");
            }
        }
}
```

2. Application:

a. Conversion from Infix to Postfix.

POSTFIX (Q,P)

- **Q** is an arithmetic expression in infix notation.
- **P** is an arithmetic expression in postfix notation.

Step 1: Start.

Step 2: Add '(' at the beginning and ')' at the end of Q.

Step 3: Scan Q from left to right and repeat step 4 to 7 until all scan is completed. i.e., stack is empty.

Step 4: If an operand is encountered, add it to P.

Step 5: If a left parenthesis is encountered, push it onto stack.

Step 6: If an operator **Θ** is encountered, check TOS.

- If TOS contains left parenthesis i.e., '(' or an operator with lower precedence, push the operator Θ onto stack.
- If TOS contains an operator with same or higher precedence than Θ; repeatedly pop from stack the operators and add to P. Push Θ onto stack.

Step 7: If a right parenthesis i.e., ')' is encountered then,

- Repeatedly pop from stack and add it to P each other until left parenthesis is encountered.
- Remove the left parenthesis.

Step 8: Stop.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
#define size 50
void push(char);
char pop();
void convert();
int precedence(char);
int tos=-1, stack[size], len;
char p[50], str[50];
```

```
void main()
{
	int i;
	char strcp[50];
	clrscr();
	printf("\n\nEnter expression :- ");
	scanf("%s",strcp);
	len=strlen(strcp);
	for(i=len-1;i>=0;i--)
	{
		strcp[i+1]=strcp[i];
	}
	strcp[0]='('; strcp[len+1]=')';
	strcp[len+2]='\0';
	strcpy(str,strcp);
	printf("\n\n\tINFIX = %s",strcp);
	convert();
	printf("\n\n\tPOSTFIX = %s",p);
	getch();
}

void push(char a)
{
	tos++;
	stack[tos]=a;
}

char pop()
{
	char a;
	a=stack[tos];
	tos--;
	return(a);
}

int precedence(char a)
{
	if (a=='-'||a=='+')
	{
		return(1);
	}
```

```
        else if (a=='*'||a=='/')
        {
                return(2);
        }
        else if (a=='$'||a=='^')
        {
                return(3);
        }
        else
        {
                return(0);
        }
}

void convert()
{
        char extra;
        int i,j=0;
        for(i=0;i<len+2;i++)        //len+2 for '(' and ')'
        {
                if(isalpha(str[i])||isdigit(str[i]))
                {
                        p[j]=str[i];
                        j++;
                }
                else if(str[i]=='(')
                {
                        push(str[i]);
                }
                else if(str[i]=='^'||str[i]=='$'||str[i]=='/'||str[i]=='*'||str[i]=='+'||str[i]=='-')
                {
                        if(stack[tos]=='('||(precedence(stack[tos])<precedence(str[i])))
                        {
                                push(str[i]);
                        }
                        else if(precedence(stack[tos])>=precedence(str[i]))
                        {
                                while(precedence(stack[tos])>=precedence(str[i]))
                                {
                                        p[j]=pop();
                                        j++;
                                }
```

```
                    push(str[i]);
                }
            }
            else if(str[i]==')')
            {
                while(stack[tos]!='(')
                {
                    p[j]=pop();
                    j++;
                }
                if(stack[tos]=='(')
                {
                    tos--;
                }
            }
        }
}
```

b. Evaluation of Postfix Expression.

Step 1: Start.
Step 2: Scan the given postfix expression P from left to right.
Repeat step 2 and step 3 until all elements are scanned.
Step 3: If the scanned element is operand, push it onto stack.
Step 4: If the scanned element is an operator **Ɵ** then,

- Pop top two elements from stack (A and B).
- Evaluate expression as A Ɵ B.
- Push the result onto stack.

Step 5: Display the TOS as final result after all elements are scanned.
Step 6: Stop.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#define max 20
int calc(int, int, char);
void push(int);
int pop();
char pfx[50];
int stack[50], tos=-1;
```

```
void main()
{
	int len,i,a,b,res,ip1;
	clrscr();
	printf("\n\nEnter the expression :- ");
	scanf("%[^\n]s",pfx);
	len=strlen(pfx);
	printf("\n\n");
	for(i=0;i<len;i++)
	{
		if(isalpha(pfx[i]))
		{
			printf("Enter value of %c :- ",pfx[i]);
			scanf("%d",&ip1);
			push(ip1);
		}
		else
		{
			a=pop(); b=pop();
			res=calc(b,a,pfx[i]);
			push(res);

		}
	}

	printf("\n\n\tResult = %d",stack[tos]);
	getch();
}

void push(int a)
{
	tos++;
	stack[tos]=a;
}

int pop()
{
	int c;
	c=stack[tos];
	tos--;
	return (c);
}
```

```
int calc(int x, int y, char op)
{
	if (op=='^')
	{
		return (pow(x,y));
	}
	else if (op=='/')
	{
		return (x/y);
	}
	else if (op=='*')
	{
		return (x*y);
	}
	else if (op=='+')
	{
		return (x+y);
	}
	else if (op=='-')
	{
		return (x-y);
	}
}
```

Queue

1. Linear Queue:

a. Head and tail varying.

Step 1: Start.

Step 2: Declare and Initialize necessary variables.

- **front = 0**; from which dequeue is done.
- **rear = -1**; from which enqueue is done.
- **MAXSIZE**; a constant for maximum size the queue can hold.
- **queue[]**; an array variable with limit of MAXSIZE.

Step 3: For **ENQUEUE** operation;

```
If rear greater than maximum size of queue
        print “Queue is FULL”
Else
        Read data from user to be enqueued
        Increase rear
        Place the data ate rear of queue
```

Step 4: For **DEQUEUE** operation;

```
If front is greater than rear
        print “Queue is EMPTY”
Else
        Dequeue data from front of queue
        Decrease front
        Display the dequeued data
```

Step 5: For **DISPLAY** operation;

```
If front is greater than rear
        print “Queue is EMPTY”
Else
        Print all i^th data of queue
        If rear greater than maximum size of queue
                print “Queue is FULL"
```

Step 6: Stop.

```
#include<stdio.h>
#include<conio.h>
#define maxsize 3
void enq();
void deq();
void dis();
int front,rear,queue[maxsize];
```

```
void main()
{
	int ch;
	front=0;
	rear=-1;
	while(1)
	{
		clrscr();
		printf("\n\nQueue:\n\n\t1. ENQUEUE\n\n\t2. DEQUEUE\n\n\t3. DISPLAY\n\n\t4.
EXIT");
		printf("\n-------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				enq();
				break;
			case  2:
				deq();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}



void enq()
{
	int data;
	if(rear>=maxsize-1)
	{

	}                                                                    else
                                                                         {
```

```
printf("\n\tQueue is
FULL");
```

```
                printf("\n\tEnter data :- ");
                scanf("%d",&data);
                rear++;
                queue[rear]=data;
                printf("\n\tDATA ENQUEUED");
        }
}

void deq()
{
        int data;
        if(front>rear)
        {
                printf("\n\tQueue is EMPTY");
        }
        else
        {
                data=queue[front];
                front++;
                printf("\n\tData = %d",data);
                printf("\n\n\tDATA DEQUEUED");
        }
}


void dis()
{
        int i;
        if(front>rear)
        {
                printf("\n\tQueue is EMPTY");
        }
        else
        {
                printf("\n(FRONT)\t\t");
                for(i=front;i<=rear;i++)
                {
                        printf("%d\t",queue[i]);
                }
                printf("(REAR)");
                if(rear>=maxsize-1)
                {
                        printf("\n\n\tQueue FULL");
```

```
            }
        }
}
```

b. Head fixed and tail varying.

Step 1: Start.

Step 2: Declare and Initialize necessary variables.

- **front = 0**; from which dequeue is done.
- **rear = -1**; from which enqueue is done.
- **MAXSIZE**; a constant for maximum size the queue can hold.
- **queue[]**; an array variable with limit of MAXSIZE.

Step 3: For **ENQUEUE** operation;

```
If rear greater than maximum size of queue
        print “Queue is FULL”
Else
        Read data from user to be enqueued
        Increase rear
        Place the data ate rear of queue
```

Step 4: For **DEQUEUE** operation;

```
If rear equals its initial value
        print “Queue is EMPTY”
Else
        Dequeue data from front of queue
        Shift all the present data to its respective bottom index
        Decrease rear
        Display the dequeued data
```

Step 5: For **DISPLAY** operation;

```
If rear equals its initial value
        print “Queue is EMPTY”
Else
        Print all i^th data of queue
        If rear greater than maximum size of queue
                print “Queue is FULL"
```

Step 6: Stop.

```
#include<stdio.h>
#include<conio.h>
#define maxsize 3
void enq();
void deq();
void dis();
```

```
int front,rear,queue[maxsize];

void main()
{
	int ch;
	front=0;
	rear=-1;
	while(1)
	{
		clrscr();
		printf("\n\nQueue:\n\n\t1. ENQUEUE\n\n\t2. DEQUEUE\n\n\t3. DISPLAY\n\n\t4.
EXIT");
		printf("\n-------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				enq();
				break;
			case  2:
				deq();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}



void enq()
{
	int data;
	if(rear>=maxsize-1)
	{
```

```
        printf("\n\tQueue is FULL");
}
```

```
	else
	{
		printf("\n\tEnter data :- ");
		scanf("%d",&data);
		rear++;
		queue[rear]=data;
		printf("\n\tDATA ENQUEUED");
	}
}


void deq()
{
	int i,data;
	if(rear==-1)
	{
		printf("\n\tQueue is EMPTY");
	}
	else
	{
		data=queue[front];
		for(i=0;i<=rear;i++)
		{
			queue[i]=queue[i+1];
		}
		rear--;

		printf("\n\tData = %d",data);
		printf("\n\n\tDATA DEQUEUED");
	}
}

void dis()
{
	int i;
	if(rear==-1)
	{
		printf("\n\tQueue is EMPTY");
	}
	else
	{
		printf("\n(FRONT)\t\t");
		for(i=front;i<=rear;i++)
```

{

```
                printf("%d\t",queue[i]);
            }
            printf("(REAR)");
            if(rear>=maxsize-1)
            {
                printf("\n\n\tQueue FULL");
            }
        }
}
```

2. Circular Queue.

Step 1: Start.

Step 2: Declare and Initialize necessary variables.

- **head = 0**; from which dequeue is done, i.e. equivalent to front.
- **tail = 0**; from which enqueue is done, i.e. equivalent to rear.
- **MAXSIZE**; a constant for maximum size the queue can hold.
- **queue[]**; an array variable with limit of MAXSIZE.

Step 3: For **ENQUEUE** operation;

```
If head equals one index above tail
    print “Queue is FULL”
Else
    Read data from user to be enqueued
    Enqueue data to the tail of queue
    Increase tail
```

Step 4: For **DEQUEUE** operation;

```
If head equals tail
    print “Queue is EMPTY”
Else
    Dequeue data from head of queue
    Increase head
    Display the dequeued data
```

Step 5: For **DISPLAY** operation;

```
If head equals tail
    print “Queue is EMPTY”
Else
    Print all i^th data of queue
    If head equals one index above tail
        print “Queue is FULL"
```

Step 6: Stop.

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

```
#include<conio.h>
#define maxsize 3
void enq();
void deq();
void dis();
int head,tail,queue[maxsize];//head means front and tail means rear

void main()
{
	int ch;
	head=0;
	tail=0;
	while(1)
	{
		clrscr();
		printf("\n\nQueue:\n\n\t1. ENQUEUE\n\n\t2. DEQUEUE\n\n\t3. DISPLAY\n\n\t4.
EXIT");
		printf("\n-------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				enq();
				break;
			case  2:
				deq();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}
```

```
void enq()
{
```

```
	int data;
	if(head==(tail+1)%maxsize)
	{
		printf("\n\tQueue is FULL");
	}
	else
	{
		printf("\n\tEnter data :- ");
		scanf("%d",&data);
		queue[tail]=data;
		tail=(tail+1)%maxsize;
		printf("\n\tDATA ENQUEUED");
	}
}

void deq()
{
	int data;
	if(head==tail)
	{
		printf("\n\tQueue is EMPTY");
	}
	else
	{
		data=queue[head];
		head=(head+1)%maxsize;
		printf("\n\tData = %d",data);
		printf("\n\n\tDATA DEQUEUED");
	}
}

void dis()
{
	int i;
	if(head==tail)
	{
		printf("\n\tQueue is EMPTY");
	}
	else
	{
		i=head;
```

```
printf("\n(HE
AD)\t\t");
```

```
                while(i!=tail)
                {
                        printf("%d\t",queue[i]);
                        i=(i+1)%maxsize;
                }
                printf("(TAIL)");
                if(head==(tail+1)%maxsize)
                {
                        printf("\n\n\tQueue is FULL");
                }
        }
}
```

Recursion and Tower of Hanoi

1. WAP for:

a. Sum of two integers (a + b).

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

void main()
{
        int a,b;
        clrscr();
        printf("Enter two numbers :- ");
        scanf("%d%d",&a,&b);
        printf("Result = %d",calc(a,b));
        getch();
}

int calc(int x, int y)
{
        if(y==0)
        {
                return x;
        }
        else
        {
                return 1+calc(x,y-1);
        }
}
```

b. Exponential value (x^y).

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

void main()
{
        int a,x,y;
        clrscr();
        printf("Enter two numbers :- ");
        scanf("%d%d",&x,&y);
```

```
	printf("Result = %d",calc(x,y));
	getch();
}

int calc (int a, int b)
{
	if(b==1)
	{
		return a;
	}
	else
	{
		return a*calc(a,b-1);
	}
}
```

c. Sum of n natural numbers.

```
#include<stdio.h>
#include<conio.h>
int calc(int);

void main()
{
	int a;
	clrscr();
	printf("Enter a number :- ");
	scanf("%d",&a);
	printf("Result = %d",calc(a));
	getch();
}

int calc(int n)
{
	if(n==1)
	{
		return n;
	}
	else
	{
		return n+calc(n-1);
	}
```

```
}
```

d. Fibonacci series.

```
#include <stdio.h>
#include <conio.h>
int fibonacci(int);

void main()
{
        int terms, counter;
        clrscr();
        printf("Enter number of terms in Fibonacci series :- ");
        scanf("%d", &terms);
        printf("\n\nFibonacci series till %d terms\n\n\t", terms);
        for(counter = 0; counter < terms; counter++)
        {
        printf("%d,\t", fibonacci(counter));
        }
        getch();
}

int fibonacci(int term)
{
        if(term < 2)
        {
                return term;
        }
        else
        {
                return fibonacci(term - 1) + fibonacci(term - 2);
        }
}
```

2. TOH.

To move **n** disk from **peg A** to **peg C** using **peg B** as auxiliary;

Step 1: Start.

Step 2: Declare and initialize necessary variables

- **n**; number of disks
- **A** = 'peg A'
- **B** = 'peg B'
- **C** = 'peg C'

Step 3: If n = 1, move the single disk from A to C and stop.
Step 4: Move the top (n-1) disk from A to B using C.
Step 5: Move the remaining disk from A to C.
Step 6: Move the (n-1) disk from B to C using A.
Step 7: Stop.

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

void main()
{
        int n;
        char a,b,c;
        clrscr();
        a='A';
        b='B';
        c='C';
        printf("\n\nEnter the value of n :- ");
        scanf("%d",&n);
        printf("\n\nResult = \n\n");
        toh(n,a,c,b);
        getch();
}

void toh(int n, char a, char c, char b)
{
        if (n==1)
        {
                printf("\tMove %d from %c to %c\n",n,a,c);
        }
        else
        {
                toh(n-1,a,b,c);
                printf("\tMove %d from %c to %c\n",n,a,c);
                toh(n-1,b,c,a);
        }
}
```

Static Linked List

1. Implementation of Static Linear Linked List.

Step 1: Start.

Step 2: Declare and initialize necessary variables and functions

- **MAXSIZE**; a constant for maximum size the list can hold
- **node[]**; a array structure with data variable and address pointer
- **avail = 0**; a variable to point the current available memory location
- **head = -1**; a variable to point the head node
- **getnode()**; a function that allocates node
- **freenode()**;a function that releases allocated node

Step 3: For **INSERT** operation;

Get a node 'n'
Assign *data* to the node 'n'
To insert this node 'n' between 'n1' → 'n2';
Point n → n2
Point n1 → n

Step 4: For **DELETE** operation;

To delete node 'n' from 'n1' → 'n' →'n2';
Point n1 → n2
Display *data* of node 'n'
Release node 'n'

Step 5: For **DISPLAY** operation;

Display data from head node till end node

Step 6: Stop.

```
#include<stdio.h>
#include<conio.h>
#define MAXSIZE 10

struct nodetype
{
        int info, next;
}node[MAXSIZE];
int avail=0, head=-1;

int getnode()
{
        int p;
        if(avail==-1)
        {
                p=avail;
        }
```

```
	else
	{
		p=avail;
		avail=node[avail].next;
	}

	return p;
}

void freenode(int p)
{
	if(p>=0 && p<=MAXSIZE-1)
	{
		node[p].next=avail;
		avail=p;
	}
	else
	{
		printf("\n\n\t\tInvalid Deletion !!! ");

	}
}

void ins()
{
	int np, p, ch, key, item, flag=0, i, temp;
	np=getnode();
	if(np==-1)
	{
		printf("\n\n\t\tMemory Cannot be Allocated. ( OVERFLOW )");
	}
	else
	{
		printf("\n\nEnter data :- ");
		scanf("%d",&item);
		node[np].info=item;
		node[np].next=-1;
		if(head==-1)
		{
			head=np;
		}
		else
		{
			clrscr();
			printf("\n\nINSERT WHERE:\n\n\t1. FRONT\n\n\t2. LAST\n\n\t3. AFTER\n\n\t4.
BEFORE");
			printf("\n--------------------------------");
			printf("\n\nSelect Location :- ");
			scanf("%d",&ch);
```

```
switch(ch)
{
	case 1:
		node[np].next=head;
		head=np;
		break;
	case 2:
		p=head;
		while(node[p].next!=-1)
		{
			p=node[p].next;
		}
		node[p].next=np;
		break;
	case 3:
		printf("\n\n\tEnter nth key :- ");
		scanf("%d",&key);
		p=head;
		for(i=0;i<key-1;i++)
		{
			if(p!=-1)
				p=node[p].next;
		}
		if(p==-1)
		{
			printf("\n\n\tINVALID INSERTION.");
			freenode(np);
			flag=1;
		}
		else
		{
			node[np].next=node[p].next;
			node[p].next=np;
		}
		break;
	case 4:

		printf("\n\n\tEnter nth key :- ");
		scanf("%d",&key);
		p=head;
		for(i=0;i<key-1;i++)
		{
			if(p!=-1)
			{
				temp=p;
				p=node[p].next;
			}
		}
		if(p==-1)
```

```
                                        {
                                                printf("\n\n\tINVALID INSERTION.");
                                                freenode(np);
                                                flag=1;
                                        }
                                        else if(key==1)
                                        {
                                                node[np].next=head;
                                                head=np;
                                        }
                                        else
                                        {
                                                node[np].next=node[temp].next;
                                                node[temp].next=np;
                                        }
                                        break;
                                default:

                                        printf("\n\n\tEnter correct choice.....");
                                        freenode(np);
                                        flag=1;
                        }
                        if(flag!=1)

                                printf("\n\n\tDATA INSERTED !!!");
                }
        }
}

void del()
{
        int p, ch, key, item, i, flag=0, temp;
        p=head;
        if(head==-1)
        {
                printf("\n\n\tList is EMPTY !!! ");
        }
        else
        {
                clrscr();
                printf("\n\nDELETE WHERE:\n\n\t1. FRONT\n\n\t2. LAST\n\n\t3. AFTER\n\n\t4.
BEFORE");
                printf("\n--------------------------------");
                printf("\n\nSelect Location :- ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                head=node[p].next;
```

```
                item=node[p].info;
                freenode(p);
                break;
        case 2:
                if(node[p].next==-1)
                {
                        head=-1;
                }
                else
                {
                        while(node[p].next!=-1)
                        {
                                temp=p;
                                p=node[p].next;
                        }
                        node[temp].next=-1;
                }
                item=node[p].info;
                freenode(p);
                break;
        case 3:
                printf("\n\n\tEnter nth key :- ");
                scanf("%d",&key);
                p=head;
                for(i=0;i<key;i++)
                {
                        if(p!=-1)
                        {
                                temp=p;
                                p=node[p].next;
                        }
                }
                if(p==-1)
                {
                        printf("\n\n\tINVALID DELETION.");
                        flag=1;
                }
                else
                {
                        node[temp].next=node[p].next;
                        item=node[p].info;
                        freenode(p);
                }
                break;
        case 4:
                printf("\n\n\tEnter nth key :- ");
                scanf("%d",&key);
                p=head;
```

```
                                for(i=0;i<key-2;i++)
                                {
                                        if(p!=-1)
                                        {
                                                temp=p;
                                                p=node[p].next;
                                        }
                                }
                                if(key==1 || node[p].next==-1)
                                {
                                        printf("\n\n\tINVALID DELETION.");
                                        flag=1;
                                }
                                else if(key==2)
                                {
                                        head=node[p].next;
                                        item=node[p].info;
                                        freenode(p);
                                }
                                else
                                {
                                        node[temp].next=node[p].next;
                                        item=node[p].info;
                                        freenode(p);
                                }
                                break;
                        default:

                                printf("\n\n\tEnter correct choice.....");
                                flag=1;
                }
                if(flag!=1)

                        printf("\n\n\tDATA DELETED = %d. !!!",item);
        }
}

void dis()
{
        int ch, p, i;
        clrscr();
        printf("\n\nDISPLAY HOW:\n\n\t1. LINEAR\n\n\t2. ARRAY");
        printf("\n------------------------------");
        printf("\n\nSelect Choice :- ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                        p=head;
```

```
if(head==-1)
```

```
                {
                        printf("\n\n\tList is EMPTY !!! ");
                }
                else
                {
                        printf("\n\nData On The List :-");
                        printf("\n----------\n\n");
                        printf(" ( HEAD ) -->  ");
                        while(p!=-1)
                        {
                                printf("%d  ->  ",node[p].info);
                                p=node[p].next;
                        }
                        printf("( NULL ) ");
                }
                break;
        case 2:

                p=head;
                if(head==-1)
                {
                        printf("\n\n\tList is EMPTY !!! ");
                }
                else
                {
                        printf("\n\n\tINDEX\tINFO\tNEXT\n");
                        printf("\t----------\n");
                        for(i=0;i<MAXSIZE;i++)
                        {
                                printf("\t %d\t %d\t %d",i,node[i].info,node[i].next);
                                if(i==p)
                                        printf("\t( HEAD )");
                                printf("\n");
                        }
                }
                break;
        default:
                printf("\n\n\tEnter correct choice.....");
    }
}

void main()
{
    int i, ch; clrscr();
    for(i=0;i<MAXSIZE-1;i++)
    {
        node[i].next=i+1;
    }
```

```
	node[MAXSIZE-1].next=-1;
	while(1)
	{
		clrscr();
		printf("\n\nStatic Linear Linked List:\n\n\t1. INSERT\n\n\t2. DELETE\n\n\t3.
DISPLAY\n\n\t4. EXIT");
		printf("\n-------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
			switch(ch)
		{
			case  1:
				ins();
				break;
			case  2:
				del();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}
```

2. Static Linear Linked List as Queue.

Same code as of Implementation of Static Linear Linked List as done above.

(ENQUEUE)

In void ins() :-

Don’t ask for choice.
Use Insert at LAST i.e.,
Use Case 2 only.
Ignore Case 1, Case 3, Case 4 and default.

(DEQUEUE)

In void del() :-

Don’t ask for choice.
Use Delete at FRONT i.e.,
Use Case 1 only.
Ignore Case 2, Case 3, Case 4 and default.

Dynamic Singly Linked List

1. Linear Dynamic Singly Linked List.

Step 1: Start.

Step 2: Declare and initialize necessary variables and functions

- **node**; a structure with data variable and address pointer
- **head**; a pointer variable to point the head node
- **getnode()**; a function that allocates memory

Step 3: For **INSERT** operation;

Get a node 'n'

Assign *data* to the node 'n'

To insert this node 'n' between 'n1' → 'n2';

Point n → n2

Point n1 → n

Step 4: For **DELETE** operation;

To delete node 'n' from 'n1' → 'n' →'n2';

Point n1 → n2

Display *data* of node 'n'

Release node 'n'

Step 5: For **DISPLAY** operation;

Display data from head node till end node

Step 6: Stop.

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

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

struct node *getnode()
{
        struct node *p;
        p=(struct node *) malloc(sizeof(struct node));
        if(p==NULL)
        {
                printf("\n\n\t\tMemory Cannot be Allocated !!! ");
        }
        else
        {
```

```
                printf("\n\nEnter data :- ");
                scanf("%d",&p->info);
                p->next=NULL;
        }
        return p;
}

void ins()
{
        int ch, key, flag=0, i;
        struct node *np, *p, *temp;
        np=getnode();
        if(head==NULL)
        {
                head=np;
        }
        else
        {
                clrscr();
                printf("\n\nINSERT WHERE:\n\n\t1. FRONT\n\n\t2. LAST\n\n\t3. AFTER\n\n\t4.
BEFORE");
                printf("\n--------------------------------");
                printf("\n\nSelect Location :- ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                np->next=head;
                                head=np;
                                break;
                        case 2:
                                p=head;
                                while(p->next!=NULL)
                                {
                                        p=p->next;
                                }
                                p->next=np;
                                break;
                        case 3:
                                printf("\n\n\tEnter nth key :- ");
                                scanf("%d",&key);
                                p=head;
                                for(i=0;i<key-1;i++)
                                {
                                        p=p->next;
                                }
                                if(p==NULL)
                                {
```

```
                                        printf("\n\n\tINVALID INSERTION.");
                                        flag=1;
                                }
                                else
                                {
                                        np->next=p->next;
                                        p->next=np;
                                }
                                break;
                        case 4:

                                printf("\n\n\tEnter nth key :- ");
                                scanf("%d",&key);
                                p=head;
                                for(i=0;i<key-1;i++)
                                {
                                        temp=p;
                                        p=p->next;
                                }
                                if(p==NULL)
                                {
                                        printf("\n\n\tINVALID INSERTION.");
                                        flag=1;
                                }
                                else if(key==1)
                                {
                                        np->next=head;
                                        head=np;
                                }
                                else
                                {
                                        np->next=temp->next;
                                        temp->next=np;
                                }
                                break;
                        default:

                                printf("\n\n\tEnter correct choice.....");
                                free(np);
                                flag=1;
                }
                if(flag!=1)

                        printf("\n\n\tDATA INSERTED !!!");
        }
}

void del()
{
        int  ch, key, item, i, flag=0;
```

```
struct node *p, *temp;
```

```
	p= head;
	if(head==NULL)
	{
		printf("\n\n\tList is EMPTY !!! ");
	}
	else
	{
		clrscr();
		printf("\n\nDELETE WHERE:\n\n\t1. FRONT\n\n\t2. LAST\n\n\t3. AFTER\n\n\t4.
BEFORE");
		printf("\n--------------------------------");
		printf("\n\nSelect Location :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				head=p->next;
				item=p->info;
				free(p);
				break;
			case 2:
				if(p->next==NULL)
				{
					head=NULL;
				}
				else
				{
					while(p->next!=NULL)
					{
						temp=p;
						p=p->next;
					}
					temp->next=NULL;
				}
				item=p->info;
				free(p);
				break;
			case 3:
				printf("\n\n\tEnter nth key :- ");
				scanf("%d",&key);
				p=head;
				for(i=0;i<key;i++)
				{
					temp=p;
					p=p->next;
				}
				if(p==NULL)
				{
```

```
                                printf("\n\n\tINVALID DELETION.");
                                flag=1;
                        }
                        else
                        {
                                temp->next=p->next;
                                item=p->info;
                                free(p);
                        }
                        break;
                case 4:

                        printf("\n\n\tEnter nth key :- ");
                        scanf("%d",&key);
                        p=head;
                        for(i=0;i<key-2;i++)
                        {
                                temp=p;
                                p=p->next;
                        }
                        if(key==1 || p->next==NULL)
                        {
                                printf("\n\n\tINVALID DELETION.");
                                flag=1;
                        }
                        else if(key==2)
                        {
                                head=p->next;
                                item=p->info;
                                free(p);
                        }
                        else
                        {
                                temp->next=p->next;
                                item=p->info;
                                free(p);
                        }
                        break;
                default:

                        printf("\n\n\tEnter correct choice.....");
                        flag=1;
        }
        if(flag!=1)

                printf("\n\n\tDATA DELETED = %d. !!!",item);
    }
}

void dis()
```

{

```
	struct node *p;
	p=head;
	if(head==NULL)
	{
		printf("\n\n\tList is EMPTY !!! ");
	}
	else
	{
		printf("\n\nData On The List :-");
		printf("\n--------------------\n\n");
		printf(" ( HEAD ) -->  ");
		while(p!=NULL)
		{
			printf("%d  ->  ",p->info);
			p=p->next;
		}
		printf("( NULL ) ");
	}
}

void main()
{
	int i, ch;
	clrscr();
	while(1)
	{
		clrscr();
		printf("\n\nDynamic Singly Linear Linked List:\n\n\t1. INSERT\n\n\t2. DELETE\n\n\t3.
DISPLAY\n\n\t4. EXIT");
		printf("\n----------------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				ins();
				break;
			case  2:
				del();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
```

```
                getch();
        }
}
```

2. Circular Dynamic Singly Linked List.

Step 1: Start.
Step 2: Declare and initialize necessary variables and functions
- **node**; a structure with data variable and address pointer
- **head**; a pointer variable to point a position for entry and deletion of data
- **getnode()**; a function that allocates memory

Step 3: For **INSERT** operation;
Get a node 'n'
Assign *data* to the node 'n'
To insert this node 'n' between 'n1' → 'n2';
Point n → n2
Point n1 → n

Step 4: For **DELETE** operation;
To delete node 'n' from 'n1' → 'n' →'n2';
Point n1 → n2
Display *data* of node 'n'
Release node 'n'

Step 5: For **DISPLAY** operation;
Display data from head node till head after a circular loop

Step 6: Stop.
[*NOTE:* Always tail node must point to head node and head node must point to tail node.]
[(tail → head)]

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

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

struct node *getnode()
{
        struct node *p;
        p=(struct node *) malloc(sizeof(struct node));
        if(p==NULL)
        {
```

```
                printf("\n\n\t\tMemory Cannot be Allocated !!! ");
        }
        else
        {
                printf("\n\nEnter data :- ");
                scanf("%d",&p->info);
                p->next=NULL;
        }

        return p;
}

void ins()
{
        struct node *np, *p;
        np=getnode();
        if(head==NULL)
        {
                head=np;
                head->next=head;
        }
        else
        {
                p=head;
                while(p->next!=head)
                        p=p->next;
                p->next=np;
                np->next=head;
                head=np;
        }

        printf("\n\n\tDATA INSERTED !!!");
}

void del()
{
        int item;
        struct node *p, *temp;
        if(head==NULL)
        {
                printf("\n\n\tList is EMPTY !!! ");
        }
        else
        {
                temp=head;
                if(temp->next==head)
                {
```

```
item=temp->info;
head=NULL;
free(temp);
```

```
                }
                else
                {
                        item=temp->info;
                        p=head;
                        while(p->next!=head)
                                p=p->next;
                        p->next=temp->next;
                        head=temp->next;
                        free(temp);
                }

                printf("\n\n\tDATA DELETED = %d. !!!",item);
        }
}

void dis()
{
        int i;
        struct node *p;
        if(head==NULL)
        {
                printf("\n\n\tList is EMPTY !!! ");
        }
        else
        {
                p=head;
                printf("\n\nData On The List :-");
                printf("\n--------------------\n\n");
                printf(" ( HEAD ) -->  ");
                for(i=0;i<2;i++)
                {
                        while(p->next!=head)
                        {
                                printf("%d  ->  ",p->info);
                                p=p->next;
                        }
                        printf("%d  ->  ",p->info);
                        p=p->next;
                }
                printf(" ..... ");
        }
}

void main()
{
        int i, ch;
        clrscr();
        while(1)
```

```
	{
		clrscr();
		printf("\n\nDynamic Singly Circular Linked List:\n\n\t1. INSERT\n\n\t2. DELETE\n\n\t3.
DISPLAY\n\n\t4. EXIT");
		printf("\n----------------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				ins();		//at head
				break;
			case  2:
				del();		//at head
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}
```

Dynamic Doubly Linked List

1. Linear Dynamic Doubly Linked List.

Step 1: Start.

Step 2: Declare and initialize necessary variables and functions

- **node**; a structure with data variable and two address pointers
- **head**; a pointer variable to point the head node
- **tail**; a pointer variable to point the tail node
- **getnode()**; a function that allocates memory

Step 3: For **INSERT** operation;

Get a node 'n'
Assign *data* to the node 'n'
To insert this node 'n' between 'n1' ←→ 'n2';
Point n ←→ n2
Point n1 ←→ n

Step 4: For **DELETE** operation;

To delete node 'n' from 'n1' ←→ 'n' ←→'n2';
Point n1 ←→ n2
Display *data* of node 'n'
Release node 'n'

Step 5: For **DISPLAY** operation;

Display data from head node till tail node
Display data from tail node till head node

Step 6: Stop.

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

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

struct node *getnode()
{
        struct node *p;
        p=(struct node *) malloc(sizeof(struct node));
        if(p==NULL)
```

```
	{
		printf("\n\n\t\tMemory Cannot be Allocated !!! ");
	}
	else
	{
		printf("\n\nEnter data :- ");
		scanf("%d",&p->info);
		p->prev=NULL;
		p->next=NULL;
	}
	return p;
}

void ins()
{
	int ch, key, flag=0, i;
	struct node *np, *p;
	np=getnode();
	if(head==NULL)
	{
		head=np;
		tail=np;
	}
	else
	{
		clrscr();
		printf("\n\nINSERT WHERE:\n\n\t1. FRONT\n\n\t2. LAST\n\n\t3. AFTER\n\n\t4.
BEFORE");
		printf("\n--------------------------------");
		printf("\n\nSelect Location :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				head->prev=np;
				np->next=head;
				head=np;
				break;
			case  2:
				tail->next=np;
				np->prev=tail;
				tail=np;
				break;
			case 3:
				printf("\n\n\tEnter nth key :- ");
				scanf("%d",&key);
				p=head;
				for(i=0;i<key-1;i++)
```

```
        {
                p=p->next;
        }
        if(p==NULL)
        {
                printf("\n\n\tINVALID INSERTION.");
                flag=1;
        }
        else if(p==tail)
        {
                tail->next=np;
                np->prev=tail;
                tail=np;
        }
        else
        {
                np->next=p->next;
                (p->next)->prev=np;
                np->prev=p;
                p->next=np;
        }
        break;
case 4:
        printf("\n\n\tEnter nth key :- ");
        scanf("%d",&key);
        p=head;
        for(i=0;i<key-1;i++)
        {
                p=p->next;
        }
        if(p==NULL)
        {
                printf("\n\n\tINVALID INSERTION.");
                flag=1;
        }
        else if(key==1)
        {
                np->next=head;
                head->prev=np;
                head=np;
        }
        else
        {
                (p->prev)->next=np;
                np->prev=p->prev;
                np->next=p;
                p->prev=np;
        }
```

```
                    break;
                default:
                    printf("\n\n\tEnter correct choice.....");
                    flag=1;
            }

            if(flag!=1)
                printf("\n\n\tDATA INSERTED !!!");
        }
}

void del()
{
        int  ch, key, item, i, flag=0;
        struct node *p;
        if(head==NULL)
        {
            printf("\n\n\tList is EMPTY !!! ");
        }
        else
        {
                clrscr();
            printf("\n\nDELETE WHERE:\n\n\t1. FRONT\n\n\t2. LAST\n\n\t3. AFTER\n\n\t4.
BEFORE");
            printf("\n-------------------------------");
            printf("\n\nSelect Location :- ");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:
                    p= head;
                    if(head->next==NULL)
                    {
                         tail=NULL;
                    }
                    else
                    {
                            head->next->prev=NULL;
                    }
                    head=head->next;
                    item=p->info;
                    free(p);
                    break;
                case 2:
                    p=tail;
                    if(head->next==NULL)
                    {
                            head=NULL;
                    }
```

```
            else
            {
                    (tail->prev)->next=NULL;
            }

            tail=tail->prev;
            item=p->info;
            free(p);
            break;
case 3:
            printf("\n\n\tEnter nth key :- ");
            scanf("%d",&key);
            p=head;
            for(i=0;i<key;i++)
            {
                    p=p->next;
            }
            if(p==NULL)
            {
                    printf("\n\n\tINVALID DELETION.");
                    flag=1;
            }
            else if(p==tail)
            {
                    (p->prev)->next=NULL;
                    tail=p->prev;
                    item=p->info;
                    free(p);
            }
            else
            {
                    (p->prev)->next=p->next;
                    (p->next)->prev=p->prev;
                    item=p->info;
                    free(p);
            }
            break;
case 4:

            printf("\n\n\tEnter nth key :- ");
            scanf("%d",&key);
            p=head;
            for(i=0;i<key-2;i++)
            {
                    p=p->next;
            }
            if(key==1 || p->next==NULL)
            {
                    printf("\n\n\tINVALID DELETION.");
```

```
flag=1;
```

```
                                        }
                                        else if(key==2)
                                        {
                                                (p->next)->prev=NULL;
                                                head=p->next;
                                                item=p->info;
                                                free(p);
                                        }
                                        else
                                        {
                                                (p->prev)->next=p->next;
                                                (p->next)->prev=p->prev;
                                                item=p->info;
                                                free(p);
                                        }
                                        break;
                                default:

                                        printf("\n\n\tEnter correct choice.....");
                                        flag=1;
                        }
                        if(flag!=1)

                                printf("\n\n\tDATA DELETED = %d. !!!",item);
                }
}

void dis()
{
        struct node *p;
        p=head;
        if(head==NULL)
        {
                printf("\n\n\tList is EMPTY !!! ");
        }
        else
        {
                printf("\n\nData On The List :-");
                printf("\n--------------------\n\n");
                printf(" ( HEAD ) -->  ");
                while(p!=NULL)
                {
                        printf("%d  ->  ",p->info);
                        p=p->next;
                }
                printf("( NULL ) ");
                printf("  <-- ( TAIL ) ");
        }
        p=tail;
```

```
if(tail==NULL)
```

```
	{
		printf("\n\n\tList is EMPTY !!! ");
	}
	else
	{
		printf("\n\n");
		printf(" ( TAIL ) -->  ");
		while(p!=NULL)
		{
			printf("%d  ->  ",p->info);
			p=p->prev;
		}
		printf("( NULL ) ");
		printf("  <-- ( HEAD ) ");
	}
}

void main()
{
	int i, ch;
	clrscr();
	while(1)
	{
		clrscr();
		printf("\n\nDynamic Doubly Linear Linked List:\n\n\t1. INSERT\n\n\t2. DELETE\n\n\t3.
DISPLAY\n\n\t4. EXIT");
		printf("\n----------------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case  1:
				ins();
				break;
			case  2:
				del();
				break;
			case 3:
				dis();
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}
```

2. Circular Dynamic Doubly Linked List.

Step 1: Start.
Step 2: Declare and initialize necessary variables and functions

- **node**; a structure with data variable and two address pointers
- **head**; a pointer variable to point the head node
- **tail**; a pointer variable to point the tail node
- **getnode()**; a function that allocates memory

Step 3: For **INSERT** operation;

Get a node 'n'
Assign *data* to the node 'n'
To insert this node 'n' between 'n1' ←→ 'n2';
Point n ←→ n2
Point n1 ←→ n

Step 4: For **DELETE** operation;

To delete node 'n' from 'n1' ←→ 'n' ←→'n2';
Point n1 ←→ n2
Display *data* of node 'n'
Release node 'n'

Step 5: For **DISPLAY** operation;

Display data from head node till tail node
Display data from tail node till head node

Step 6: Stop.
[*NOTE:* Always tail node must point to head node and head node must point to tail node.]
[(tail → head) and (head → tail)]

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

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

struct node *getnode()
{
        struct node *p;
        p=(struct node *) malloc(sizeof(struct node));
        if(p==NULL)
        {
                printf("\n\n\t\tMemory Cannot be Allocated !!! ");
```

```
	}
	else
	{
		printf("\n\nEnter data :- ");
		scanf("%d",&p->info);
		p->prev=NULL;
		p->next=NULL;
	}

	return p;
}

void ins()
{
	struct node *np;
	np=getnode();
	if(head==NULL)
	{
		head=np;
		head->next=head;
		head->prev=head;
	}
	else
	{
		(head->prev)->next=np;
		np->prev=head->prev;
		np->next=head;
		head->prev=np;
		head=np;
	}

	printf("\n\n\tDATA INSERTED !!!");
}

void del()
{
	int item;
	struct node *temp;
	if(head==NULL)
	{
		printf("\n\n\tList is EMPTY !!! ");
	}
	else
	{
		temp=head;
		if(temp->next==head)
		{
```

```
item=temp->info;
head=NULL;
free(temp);
```

```
                }
                else
                {
                        item=temp->info;
                        (temp->prev)->next=temp->next;
                        (temp->next)->prev=temp->prev;
                        head=temp->next;
                        free(temp);
                }

                printf("\n\n\tDATA DELETED = %d. !!!",item);
        }
}

void dis()
{
        int i;
        struct node *p;
        if(head==NULL)
        {
                printf("\n\n\tList is EMPTY !!! ");
        }
        else
        {
                p=head;
                printf("\n\nData On The List :-");
                printf("\n--------------------\n\n");
                printf(" ( HEAD ) -->  ");
                for(i=0;i<2;i++)
                {
                        while(p->next!=head)
                        {
                                printf("%d  ->  ",p->info);
                                p=p->next;
                        }
                        printf("%d  ->  ",p->info);
                        p=p->next;
                }
                printf(" ........ ");
                p=head;
                printf("\n\n");
                printf(" ........ -->  ");
                for(i=0;i<2;i++)
                {
                        while(p->prev!=head)
                        {
                                p=p->prev;
                                printf("%d  ->  ",p->info);
                        }
```

```
                        p=p->prev;
                        printf("%d  ->  ",p->info);
                }
                printf("  <-- ( HEAD ) ");
        }
}

void main()
{
        int i, ch;
        clrscr();
        while(1)
        {
                clrscr();
                printf("\n\nDynamic Doubly Circular Linked List:\n\n\t1. INSERT\n\n\t2. DELETE\n\n\t3.
DISPLAY\n\n\t4. EXIT");
                printf("\n----------------------------------------");
                printf("\n\nEnter your choice :- ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case  1:
                                ins();              //at head
                                break;
                        case  2:
                                del();              //at head
                                break;
                        case 3:
                                dis();
                                break;
                        case 4:
                                exit(0);
                        default:
                                printf("\n\n\tEnter correct choice.....");
                }
                getch();
        }
}
```

Binary Search Tree (BST)

1. Basic Operations (Insert, Search) and Traversal (Pre-Order, In-Order, Post-Order) on BST.

Step 1: Start.

Step 2: For **INSERT** operation;

```
If root is NULL
        then create root node
        return
end if
If root exists then
        compare the data with node.data
        while until insertion position is located
                If data is greater than node.data
                        goto right subtree
                else
                        goto left subtree
        endwhile
        insert data
end If
```

Step 3: For **SEARCH** operation;

```
If root.data is equal to search.data
        return root
else
        while data not found
                If data is greater than node.data
                        goto right subtree
                else
                        goto left subtree
                If data found
                        return node
        endwhile
        return data not found
end if
```

Step 4: For **IN-ORDER TRAVERSAL** operation;

```
Until all nodes are traversed:
        Recursively traverse left subtree.
        Visit root node.
        Recursively traverse right subtree.
```

Step 5: For **PRE-ORDER TRAVERSAL** operation;

```
Until all nodes are traversed:
        Visit root node.
        Recursively traverse left subtree.
        Recursively traverse right subtree.
```

Step 6: For **POST-ORDER TRAVERSAL** operation;

Until all nodes are traversed:

Recursively traverse left subtree.
Recursively traverse right subtree.
Visit root node.

Step 7: Stop.

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

struct node {
        int data;
        struct node *leftChild;
        struct node *rightChild;
};
struct node *root = NULL;

void insert()
{
        struct node *tempNode = (struct node*) malloc(sizeof(struct node));
        struct  node  *current;
        struct  node  *parent;
        int data;
        printf("\n\nEnter the data :- ");
        scanf("%d",&data);
        tempNode->data = data;
        tempNode->leftChild = NULL;
        tempNode->rightChild = NULL;
        //if tree is empty
        if(root == NULL)
        {
                root = tempNode;
        }
        else
        {
                current = root;
                parent = NULL;
                while(1)
                {
                        parent = current;
                        //go to left of the tree
                        if(data < parent->data)
                        {
                                current = current->leftChild;
```

```
                        //insert to the left
                        if(current == NULL)
                        {
                                parent->leftChild = tempNode;
                                return;
                        }
                }
                else    //go to right of the tree
                {
                        current = current->rightChild;
                        //insert to the right
                        if(current == NULL)
                        {
                                parent->rightChild = tempNode;
                                return;
                        }
                }
        }
    }
}

struct node* search(int data)
{
        struct node *current = root;
        printf("\nVisiting elements: \n\n\t");
        while(current->data != data)
        {
                if(current != NULL)
                {
                        printf("%d,\t",current->data);
                }
                //go to left tree
                if(current->data > data)
                {
                        current = current->leftChild;
                }
                else      //else go to right tree
                {
                        current = current->rightChild;
                }
                //not found
                if(current == NULL)
```

```
                {
                        return NULL;
                }
        }
        return current;
}

void pre_order_traversal(struct node* root)
{
        if(root != NULL)
        {
                printf("%d ",root->data);
                pre_order_traversal(root->leftChild);
                pre_order_traversal(root->rightChild);
        }
}

void inorder_traversal(struct node* root)
{
        if(root != NULL)
        {
                inorder_traversal(root->leftChild);
                printf("%d ",root->data);
                inorder_traversal(root->rightChild);
        }
}

void post_order_traversal(struct node* root)
{
        if(root != NULL)
        {
                post_order_traversal(root->leftChild);
                post_order_traversal(root->rightChild);
                printf("%d ", root->data);
        }
}

void main()
{
        struct node * temp;
        int ch, data;
        while(1)
```

```
	{
		clrscr();
		printf("\n\nBinary Search Tree:\n\n\t1. INSERT\n\n\t2. SEARCH\n\n\t3.
TRAVERSE\n\n\t4. EXIT");
		printf("\n-------------------------------");
		printf("\n\nEnter your choice :- ");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				insert();
				printf("\n\t\tDATA INSERTED.");
				break;
			case 2:
				printf("\n\nEnter data to search :- ");
				scanf("%d",&data);
				temp = search(data);
				if(temp != NULL)
				{
					printf("[ %d ]\n\n\tElement found (%d).",temp->data,data);
				}
				else
				{
					printf("[ x ]\n\n\tElement not found (%d).", data);
				}
				break;
			case 3:
				printf("\nPreorder traversal:\n\t");
				pre_order_traversal(root);
				printf("\nInorder traversal:\n\t");
				inorder_traversal(root);
				printf("\nPost order traversal:\n\t");
				post_order_traversal(root);
				break;
			case 4:
				exit(0);
			default:
				printf("\n\n\tEnter correct choice.....");
		}
		getch();
	}
}
```

Sorting Algorithms

1. Bubble Sort.

Step 1: Start.
Step 2: Loop through all elements of list
Loop through elements falling ahead
If current element is greater than next element
Swap them to bubble up the highest element
End if
Step 3: Stop.

```
#include <stdio.h>
#include<conio.h>
#define MAX 10
int list[MAX] = {1,8,4,6,0,3,5,2,7,9};

void display()
{
	int i;
	printf("[[   ");
	for(i = 0; i < MAX; i++)
	{
		printf("%d   ",list[i]);
	}
	printf("]]");
}

void bubbleSort()
{
	int temp;
	int i,j;
	for(i = 0; i < MAX-1; i++)
	{
		printf("Iteration %d : \n",i+1);
		for(j = 0; j < MAX-1-i; j++)
		{
			printf(" Items compared: [ %d, %d ] ", list[j],list[j+1]);
			if(list[j] > list[j+1])
			{
				temp = list[j];
				list[j] = list[j+1];
				list[j+1] = temp;
```

```
                        printf(" => swapped [%d, %d]\n",list[j],list[j+1]);
                    }
                    else
                    {
                        printf(" => not swapped\n");
                    }
                }

                printf("\nAfter Iteration %d#: ",(i+1));
                display();
                printf("\n\n");
                getch();
        }
}

void main()
{
        clrscr();
        printf("\n\nInput Array: \n\n\t");
        display();
        printf("\n\n");
        getch();
        bubbleSort();
        printf("\nOutput Array: \n\n\t");
        display();
        getch();
}
```

2. Merge Sort.

Step 1: Start.
Step 2: If it is only one element in the list it is already sorted, return.
Step 3: Divide the list recursively into two halves until it can no more be divided.
Step 4: Merge the smaller lists into new list in sorted order.
Step 5: Stop.

```
#include <stdio.h>
#include<conio.h>
#define max 10
int ct_merge = 0;
int ct_divide = 0;
int a[max] = {1,8,4,6,0,3,5,2,7,9};
int b[max];
```

```
void display()
{
	int i;
	printf("[[   ");
	for(i = 0; i < max; i++)
	{
		printf("%d   ",a[i]);
	}
	printf("]]");
}

void merging(int low, int mid, int high)
{
	int l1, l2, i;
	for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++)
	{
		if(a[l1] <= a[l2])
			b[i] = a[l1++];
		else
			b[i] = a[l2++];
	}
	while(l1 <= mid)
		b[i++] = a[l1++];
	while(l2 <= high)
		b[i++] = a[l2++];
	printf("\nMerge  %d:--  ",ct_merge);
	ct_merge++;
	for(i = low; i <= high; i++)
	{
		a[i] = b[i];
		printf("%d  ",a[i]);
	}
}

void sort(int low, int high)
{
	int mid,i;
	printf("\nDivide %d:--  ",ct_divide);
	ct_divide++;
	for(i = low; i <= high; i++)
	{
		printf("%d  ",a[i]);
```

```
	}
	if(low < high)
	{
		mid = (low + high) / 2;
		sort(low, mid);
		getch();
		sort(mid+1, high);
		getch();
		merging(low, mid, high);
	}
	else
	{
		return;
	}
}


void main()
{
	clrscr();
	printf("\n\nList before sorting\n\n\t");
	display();
	printf("\n");
	sort(0, max-1);
	printf("\n\n\nList after sorting\n\n\t");
	display();
	getch();
}
```