Linear Search

```
1. /*
2. ####  LINEAR SEARCH  #####
3.
4. iterative approach #using For Loop
5.
6. ## TIME COMPLECITY ##
7. IN BEST CASE               O(1)
8. IN WROST CASE              O(n)
9. IN AVERAGE CASE     =      SUM OF ALL THE CASES / TOTAL NO OF CASES
10. IN AVERAGE CASE           O(n+1)/2
11.
12. */
13. #include<stdio.h>
14. #define N 100
15.
16. int Arr[N];
17.
18. int LinearSearch(int Arr[],int search);
19.
20. int main(){
21.       int ins,i,search,Result;
22.       printf("How many Numbers You want To insert ? ");
23.       scanf("%d",&ins);
24.       for(i=0;i<ins;i++){
25.             printf("Enter %d value ",i+1);
26.             scanf("%d",&Arr[i]);
27.       }
28.
29.       printf("Which Number's Position You want to Search ?");
30.       scanf("%d",&search);
31.
32.       Result=LinearSearch(&Arr[0],search);
33.
34.       if(Result!=0){
35.             printf("\n The Element Found At described Position is %d",Result);
36.       }
37.       return 0;
38. }
39.
40. int LinearSearch(int Arr[],int search){
41.       int i,found=0;
42.       for(i=0;i<search;i++){
43.             if(Arr[i]==search){
44.                   found=Arr[i];
45.                   printf("The Position Of Element Is %d ",i+1);
46.                   return found;
47.                   break;
48.             }
49.             else if(found==Arr[i]){
50.                   printf("\n ! Element Does Not Exist In System !");
51.                   return 0;
52.             }
53.       }
54. }
```

Binary Search

```
/*
        #### BINARY SEARCH ####

        !!!! WHILE APPLAING BINARY SEARCH DATA MUST BE SORTED !!!!
        =>BINARY SEARCH IS BASED ON DIVIED & CONQUAR RULE

        MID= L+R/2

        THREE CASES POSSIBLE
        1. DATA YOU WANT TO FIND OUT IS EQUAL TO MID POSITION
        2. DATA YOU WANT TO FIND OUT IS LESS THAN MID POSITION
        3. DATA YOU WANT TO FIND OUT IS GREATER THAN MID POSITION

//      SUPPOSE WE  HAVE AN ARRAY SIZE OF 10 INDEX 0 TO 9
//      L       R       MID
//      0       9       4       (0+9/2)
//      5       9       7       (5+9/2)
//      5       6       5       (5+6/2)
//      6       6       6       (6+6/2)

## TIME COMPLECITY ##
IN BEST CASE            O(1)
IN WROST CASE           O(log n)
IN AVERAGE CASE         O(log n)

*/
#include<stdio.h>
#define N 100

int Arr[N];

int BinarySearch(int Arr[],int search,int n);

int main(){
        int n,i,search,Result;
        printf("How many Numbers You want To insert ? ");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter %d value ",i+1);
                scanf("%d",&Arr[i]);
        }

        printf("Which Number's Position You want to Search ?");
        scanf("%d",&search);

        Result=BinarySearch(&Arr[0],search,n);

        if(Result!=0){
                printf("\n The Element Found is %d",Result);
        }
        else{
                printf("Opps..!! \nThe Element you'r Searching Does not Exist in system ");
        }
        return 0;
}

int BinarySearch(int Arr[],int search,int n){
        int i,mid,l,r;
        l=0,r=n;
        while(l<r){
                mid=(l+r)/2;
                if(search==Arr[mid]){
                        return Arr[mid];
                }
                else if(search<Arr[mid]){
                        r=mid-1;
                }
                else if(search>Arr[mid]){
                        l=mid+1;
                }
        }
        return 0;
}
```

Bubble Sort

```
1. /*
2. ####   BUBBLE SORT         ####
3.
4. TIME COMPLECITY
5. IN BEST CASE  O(n)
6. IN WROST CASE       O(n^2)
7.
8. */
9.
10. #include<stdio.h>
11. #define N 100
12.
13. int BubbleSort(int Arr[],int n);
14.
15. int main(){
16.       int n,Result,i;
17.       int Arr[N];
18.       printf("How many numbers You want to insert ? ");
19.       scanf("%d",&n);
20.       for(i=0;i<n;i++){
21.             printf("Enter %d value ",i+1);
22.             scanf("%d",&Arr[i]);
23.       }
24.
25.       Result=BubbleSort(&Arr[0],n);
26.       printf("Sorted Successfully In %d passes.\n",Result);
27.       for(i=0;i<n;i++){
28.             printf("%d ",Arr[i]);
29.       }
30.       return 0;
31. }
32.
33. //OPTIMISED BUBBLE SORT
34. int BubbleSort(int Arr[],int n){
35.       int pass,i,temp,flag;
36.
37.       for(pass=0;pass<n-1;pass++){
38.             flag=0;
39.             for(i=0;i<n-1-pass;i++){
40.                   if(Arr[i]>Arr[i+1]){
41.                         temp=Arr[i];
42.                         Arr[i]=Arr[i+1];
43.                         Arr[i+1]=temp;
44.                         flag=1;
45.                   }
46.             }
47.             if(flag==0){
48.                   break;
49.             }
50.       }
51.       return pass+1;
52. }
```

Insertion Sort

```
/*
####   INSERTION SORT      ####

IN INSERTION SORT FIRST ELEMENT WILL BE CONSIDERED AS SORTED

SORTED SUBLIST | UNSORTED SUBLIST


TIME COMPLECITY
IN BEST CASE  O(n)
IN WROST CASE O(n^2)

*/

#include<stdio.h>
#define N 100

void InsertionSort(int Arr[],int n);

int main(){
	int n,Result,i;
	int Arr[N];
	printf("How many numbers You want to insert ? ");
	scanf("%d",&n);
	for(i=0;i<n;i++){
		printf("Enter %d value ",i+1);
		scanf("%d",&Arr[i]);
	}

	InsertionSort(&Arr[0],n);
	printf("Sorted Successfully\n");
	for(i=0;i<n;i++){
		printf("%d ",Arr[i]);
	}
	return 0;
}


void InsertionSort(int Arr[],int n){
	int i,j,temp;

	for(i=1;i<n;i++){
		temp=Arr[i];
		j=i-1;
		while(j>=0 && Arr[j]>temp){
			Arr[j+1]=Arr[j];
			j--;
		}
		Arr[j+1]=temp;
	}
}
```

Selection Sort

```
1. /*
2. ####   SELECTION SORT            ####
3.
4. IN SELECTION SORT SORTED LIST IS INITIALLY EMPTY
5. SORTED SUBLIST      | UNSORTED SUBLIST
6.
7.
8. FIND MINIMUM ELEMET AND THAT ELEMENT WOULD BE SWAPPED WITH
9. THE ELEMENT WHICH IS AT THE STARTING POSITION IN THE UNSORTED SUB ARRAY
10.
11.
12.TIME COMPLECITY
13.IN BEST CASE        O(n^2)
14.IN WROST CASE       O(n^2)
15.
16.*/
17.
18.#include<stdio.h>
19.#define N 100
20.
21.void SelectionSort(int Arr[],int n);
22.
23.int main(){
24.       int n,Result,i;
25.       int Arr[N];
26.       printf("How many numbers You want to insert ? ");
27.       scanf("%d",&n);
28.       for(i=0;i<n;i++){
29.             printf("Enter %d value ",i+1);
30.             scanf("%d",&Arr[i]);
31.       }
32.
33.       SelectionSort(&Arr[0],n);
34.       printf("Sorted Successfully\n");
35.       for(i=0;i<n;i++){
36.             printf("%d ",Arr[i]);
37.       }
38.       return 0;
39.}
40.
41.
42.void SelectionSort(int Arr[],int n){
43.       int pass,i,temp;
44.       for(pass=0;pass<n-1;pass++){
45.             for(i=pass+1;i<n;i++){
46.                   if(Arr[pass]>Arr[i]){
47.                         temp=Arr[pass];
48.                         Arr[pass]=Arr[i];
49.                         Arr[i]=temp;
50.                   }
51.             }
52.       }
53.}
```

Quick Sort

```
/*
####    QUICK SORT     ####

BASED ON DIVIED AND QUANQUER TECHNICH
PARTITION IS KNOWN AS THE BACKBONE OF THE QUICK SORT


TIME COMPLECITY
IN BEST CASE   O(nlogn)
IN WROST CASE  O(n^2)

*/

#include<stdio.h>
#define N 100
#define true 1
#define false 0

void QuickSort(int Arr[],int lb,int ub);
int main(){
        int n,Result,i,lb,ub;
        int Arr[N];
        printf("How many numbers You want to insert ? ");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter %d value ",i+1);
                scanf("%d",&Arr[i]);
        }
        lb=0;
        ub=n;
        QuickSort(&Arr[0],lb,ub);
        printf("Sorted Successfully\n");
        for(i=0;i<n;i++){
                printf("%d ",Arr[i]);
        }
        return 0;
}


void QuickSort(int Arr[],int lb,int ub){
int pivot,i,j,flag,temp;
        flag=true;
        if(lb<ub){
                i=lb;
                j=ub+1;
                pivot=Arr[lb];
                while(flag){
                        i=i+1;
                        while(Arr[i]<pivot){
                                i++;
                                j--;
                        }
                        while(Arr[j]>pivot){
                                j--;
                        }
                        if(i<j){
                                temp=Arr[i];
                                Arr[i]=Arr[j];
                                Arr[j]=temp;
                        }
                        else{
                                flag=false;
                                temp=Arr[lb];
                                Arr[lb]=Arr[j];
                                Arr[j]=temp;
                                QuickSort(Arr,lb,j-1);
                                QuickSort(Arr,j+1,ub);
                        }
                }
        }
}
```

Merge Sort

```
/*
####    MERGE SORT              ####

TIME COMPLECITY
IN BEST CASE     O(nlogn)
IN WROST CASE    O(nlogn)

*/

#include<stdio.h>
#define N 100
#define true 1
#define false 0

void MergeSort(int Arr[],int lb,int ub);
void Merge(int Arr[],int lb,int mid,int ub);
int main(){
        int n,Result,i,lb,ub;
        int Arr[N];
        printf("How many numbers You want to insert ? ");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter %d value ",i+1);
                scanf("%d",&Arr[i]);
        }
        lb=0;
        ub=n;
        MergeSort(&Arr[0],lb,ub);
        printf("Sorted Successfully\n");
        for(i=0;i<n;i++){
                printf("%d ",Arr[i]);
        }
        return 0;
}


void MergeSort(int Arr[],int lb,int ub){
        int mid;
        if(lb<ub){
                mid=(lb+ub)/2;
                MergeSort(Arr,lb,mid);
                MergeSort(Arr,mid+1,ub);
                Merge(Arr,lb,mid,ub);
        }
}
void Merge(int Arr[],int lb,int mid,int ub){
        int i,j,k;
        int B[100];
        i=lb;
        j=mid+1;
        k=lb;
        while(i<=mid && j<=ub){
                if(Arr[i]<=Arr[j]){
                        B[k]=Arr[i];
                        i++;
                }
                else{
                        B[k]=Arr[j];
                        j++;
                }
                k++;
        }
        if(i>mid){
                while(j<=ub){
                        B[k]=Arr[j];
                        j++;
                        k++;
                }
        }
        else{
                while(i<=mid){
                        B[k]=Arr[i];
                        i++;
                        k++;
                }
        }
        for(k=lb;k<=ub;k++){
                Arr[k]=B[k];
        }
}
```

Heap Sort

```
1.  /*
2.  ####    HEAP SORT              ####
3.
4.  MAX HEAP
5.  FOR EVERY NODE i , THE VALUE OF NODE IS
6.  LESS THEN OR EQUAL TO ITS PARENT VALUE. except root node
7.  A[Parent[i]] >= A[i]
8.
9.  MIN HEAP
10. FOR EVERY NODE i, THE VALUE OF HEAP IS
11. GREATER THAN OR EQUAL TO ITS PARENT VALUE. except root node
12. A[Parent[i]] <= A[i]
13.
14. TIME COMPLECITY
15. IN BEST CASE   O(n)
16. IN WROST CASE  O(n^2)
17.
18. */
19.
20. #include<stdio.h>
21. #define N 100
22.
23. void HeapSort(int a[],int n);
24. void insertion(int a[],int n);
25. int main(){
26.         int n,Result,i;
27.         int a[N];
28.         printf("How many numbers You want to insert ? ");
29.         scanf("%d",&n);
30.         for(i=0;i<n;i++){
31.                 printf("Enter %d value ",i+1);
32.                 scanf("%d",&a[i]);
33.         }
34.
35.         HeapSort(&a[0],n);
36.         printf("Sorted Successfully\n");
37.         for(i=0;i<n;i++){
38.                 printf("%d ",a[i]);
39.         }
40.         return 0;
41. }
42.
43.
44. void HeapSort(int a[],int n){
45.         int i,j,k,key;
46.
47.         for(k=2;k<=n;k++){
48.                 i=k;
49.                 key=a[k];
50.                 j=i/2;
51.                 while(key>=a[j]&&i>1){
52.                         a[i]=a[j];
53.                         i=j;
54.                         j=i/2;
55.                         if(j<1){
56.                                 j=1;
57.                         }
58.                 }
59.                 a[i]=key;
60.         }
61.         insertion(&a[0],n);
62. }
```

```
63. void insertion(int *a,int n){
64.         int i,j,temp;
65.
66.         for(i=1;i<n;i++){
67.                 temp=a[i];
68.                 j=i-1;
69.                 while(j>=0 && a[j]>temp){
70.                         a[j+1]=a[j];
71.                         j--;
72.                 }
73.                 a[j+1]=temp;
74.         }
75. }
```

Shell Sort

```
1.  /*
2.  ####    SHELL SORT            ####
3.
4.
5.  TIME COMPLECITY
6.  The time complexity depends on the distance you take
7.  IN WROST CASE  O(n^2)
8.
9.  */
10.
11. #include<stdio.h>
12. #define N 100
13. #define false 0
14. #define true 1
15. void ShellSort(int Arr[],int n);
16.
17. int main(){
18.         int n,Result,i;
19.         int Arr[N];
20.         printf("How many numbers You want to insert ? ");
21.         scanf("%d",&n);
22.         for(i=0;i<n;i++){
23.                 printf("Enter %d value ",i+1);
24.                 scanf("%d",&Arr[i]);
25.         }
26.
27.         ShellSort(&Arr[0],n);
28.         printf("Sorted Successfully\n");
29.         for(i=0;i<n;i++){
30.                 printf("%d ",Arr[i]);
31.         }
32.         return 0;
33. }
34.
35.
36. void ShellSort(int Arr[],int n){
37.         int d,k,key,flag,j;
38.         for(d=n/2;d>=1;d=d/2){
39.                 for(k=d;k<n;k++){
40.                         key=Arr[k];
41.                         j=k-d;
42.                         flag=false;
43.                         while(j>=0&&flag==false){
44.                                 if(key<Arr[j]){
45.                                         Arr[j+d]=Arr[j];
46.                                         j=j-d;
47.                                 }
48.                                 else{
49.                                         flag=true;
50.                                 }
51.                         }
52.                         Arr[j+d]=key;
53.                 }
54.         }
55. }
```

Radix Sort

```
/*
####   RADIX SORT   ####

SORTING BY DIGITS
FIND MAXIMUM NUMBER
MAKE ALL NUMBERS LIKE MAXIMUM NUMBER WITHOUT CHANGING THE VALUE OF NUMBER
EX: 324,045,001

FOR NUMBERS BASE IS  10
FOR ALPHABETS BASE IS       25

TIME COMPLECITY
==>    O(d*(n+b))    <==
d=number of digits
n=how many numbers
b=base

       ### COUNTING SORT ###

       -> SORTING ACCORDING TO KEYS
       -> COUNTING THE ELEMENTS HAVING DISTINCT KEY VALUES

       TIME COMPLECITY OF COUNTING SORT
       ==>    O(n+k) <==
              n=number of elements in the array
              k=the given range
              upper limit for the k can be O(n)
              could sort will not work for negative or
              floating point vales

*/

#include<stdio.h>
#define N 100

void RadixSort(int Arr[],int n);
void CountingSort(int Arr[],int n,int pos);
int main(){
       int n,Result,i;
       int Arr[N];
       printf("How many numbers You want to insert ? ");
       scanf("%d",&n);
       for(i=0;i<n;i++){
              printf("Enter %d value ",i+1);
              scanf("%d",&Arr[i]);
       }

       RadixSort(&Arr[0],n);
       printf("Sorted Successfully\n");
       for(i=0;i<n;i++){
              printf("%d ",Arr[i]);
       }
       return 0;
}

void RadixSort(int Arr[],int n){
       int max=0,pos,i;
       for(i=0;i<n;i++){
              if(Arr[i]>max){
                     max=Arr[i];
              }
       }
       for(pos=1;max/pos>0;pos=pos*10){
              CountingSort(&Arr[0],n,pos);
       }
}
```

```
void CountingSort(int Arr[],int n,int pos){
	int i,k=10;
	int Count[k],B[n];

	for(i=0;i<k;i++){
		Count[i]=0;
	}
	for(i=0;i<n;i++){
		++Count[(Arr[i]/pos)%10];
	}
	for(i=1;i<k;i++){
		Count[i]=Count[i]+Count[i-1];
	}
	for(i=n-1;i>=0;i--){
		B[--Count[(Arr[i]/pos)%10]]=Arr[i];
	}
	for(i=0;i<n;i++){
		Arr[i]=B[i];
	}
}
```

Data Structure

Implementation Of Stack

```
/*
Definitions :
	Stack is a non-premitive data Structure
	in which Insertion and delection operations
	perform at only one end and That end is known as top.

	Stack is a last in first out data structure
	in which insertion and delection operaction
	perform at same end and That end is known as top.


	Applications On Stack

	1.Balancing of Symbols
	2.String Reversal
	3.Undo/Redo
	4.Recursion
	5.DFS(Depth First Search)
	6.Backtracking
	7.Memory Management
	8.Expression Conversion
		i. Infix to Prefix
		ii. Infix to Postfix
		iii. Prefix to infix
		iv. Postfix to infix
		v. Prefix to Postfix
	*/
```

```
#include<stdio.h>
#include<stdlib.h>
#define N 5

//FUNCTIONS
void push();
void pop();
void peep();
void change();
void Display();

//GLOBAL
int stack[N];
int top=0;

//DRIVER PROGRAM
int main(){
	int choice=0;
	while(choice!=6){
		printf("\n***STACK MENU***\n\n");
		printf("\n1. PUSH");
		printf("\n2. POP");
		printf("\n3. PEEK");
		printf("\n4. CHANGE");
		printf("\n5. DISPLAY");
		printf("\n6. EXIT");
		printf("\nENTER CHOICE ");
		scanf("%d",&choice);
			switch(choice){
				case 1:
					push();
					break;
				case 2:
					pop();
					break;
				case 3:
					peep();
					break;
				case 4:
					change();
					break;
				case 5:
					Display();
					break;
				case 6:
					exit(0);
				default:
					printf("\nERROR:-> WRONG CHOICE ");
					break;
			}
	}
	return 0;
}

//TO PUSH THE ELEMENTS IN STACK
void push(){
	int x;
	if(top==N){
		printf("\nOVERFLOW ON PUSH");
	}
	else{
		printf("ENTER X ");
		scanf("%d",&x);
		top++;
		stack[top]=x;
		Display();
	}
```

```
}

//TO POP THE FIRST ELEMENT FROM THE STACK
      void pop(){
            if(top==0){
                  printf("\n UNDERFLOW ON POP");
            }
            else{
                  printf("\n%d Deleted",stack[top]);
                  top--;
            }
      }

      //TO SEE ELEMENT AT GIVEN POSITION (LIFO)
      void peep(){
            int pos,x;
            printf("ENTER POSITION ");
            scanf("%d",&pos);
            if(top-pos+1<=0){
                  printf("\nUNDERFLOW ON PEEP");
            }
            else if(top-pos+1>N){
                  printf("\nOVERFLOW ON PEEP");
            }
            else{
                  x=stack[top-pos+1];
                  printf("%d",x);
            }
      }

      //TO CHANGE ELEMENT AT GIVEN POSITION (LIFO)
      void change(){
            int pos,x;
            printf("ENTER POSITION ");
            scanf("%d",&pos);
            if(top-pos+1<=0){
                  printf("\nUNDERFLOW ON CHANGE");
            }
            else if(top-pos+1>N){
                  printf("\nOVERFLOW ON CHANGE");
            }
            else{
                  printf("ENTER X ");
                  scanf("%d",&x);
                  stack[top-pos+1]=x;
            }
      }


      //TO DISPLAY STACK
      void Display(){
            int i;
            for(i=top;i>0;i--){
                  printf("%d ",stack[i]);
            }
      }
```

Implementation Of Queue

```
/*
Definition:
        Queue is a non-linear primitive Data Structure in which
        insertion operations perform at One end named Rear and
        deletion operations perform at another end named Front.

        Queue is a first in first out Data structure in which
        insertion operations perform at rear end and
        deletion operations perform at front end


Applications Of Queue
        1.      Queues are widely used as waiting lists for a single
                shared resource like printer, disk, CPU.
        2.      Queues are used in asynchronous transfer of data
                (where data is not being transferred at the same rate between two processes)
                for e.g. pipes, file IO, sockets.
        3.      Queues are used as buffers in most of the applications
                like MP3 media player, CD player, etc.
        4.      Queue are used to maintain the play list in media players
                in order to add and remove the songs from the play-list.
        5.      Queues are used in operating systems for handling interrupts.


TIME COMPLEXITY
FOR INSERTION AND DELECTION   O(1)
IN WROST CASE                 O(n)

Types Of Queue
  1.Linear Queue(Simple Queue)
  2.Circular Queue
  3.Priority Queue
  4.Deque
     i.  Input restricted Deque
     ii. Output restricted Deque

*/
```

Implementation Of Linear Queue

```
/*

 ### Linear Queue ###

*/

#include<stdio.h>
#include<stdlib.h>
#define N 5

//GLOBAL
int queue[N];
int front=-1,rear=-1;

//FUNCTIONS
void enque();
void deque();
void display();
void peek();

```

```
21. //DRIVER PROGRAM
22. int main(){
23.     int choice;
24.     do{
25.             printf("\n***Queue Operations***\n");
26.             printf("1. Enque\n");
27.             printf("2. Deque\n");
28.             printf("3. Peek\n");
29.             printf("4. Display\n");
30.             printf("0 To Exit\n");
31.             printf("Enter Choice");
32.             scanf("%d",&choice);
33.             switch(choice){
34.                     case 1:
35.                             enque();
36.                             break;
37.                     case 2:
38.                             deque();
39.                             break;
40.                     case 3:
41.                             peek();
42.                             break;
43.                     case 4:
44.                             display();
45.                             break;
46.                     case 0:
47.                             exit(0);
48.                             break;
49.                     default:
50.                             printf("\n!!Wrong choice!!\n");
51.             }
52.     }while(choice!=0);
53.     return 0;
54. }
55.
56. //TO INSERT AN ELEMENT
57. void enque(){
58.     int x;
59.     if(rear==N-1){
60.             printf("\noverflow\n");
61.     }
62.     else if(front==-1 && rear==-1){
63.             front++;
64.             rear++;
65.             printf("Enter x ");
66.             scanf("%d",&x);
67.             queue[rear]=x;
68.     }
69.     else{
70.             rear++;
71.             printf("Enter x ");
72.             scanf("%d",&x);
73.             queue[rear]=x;
74.     }
75. }
76.
77. //TO DELETE AN ELEMENT
78. void deque(){
79.     if(front==-1&&rear==-1){
80.             printf("\nQueue is Empty\n");
81.     }
82.     else if(front==rear){
83.             front=rear=-1;
84.             printf("\nDeleted successfully\n");
85.     }
86.     else{
87.             front++;
88.             printf("\nDeleted successfully\n");
89.     }
90. }
```

```
91. //TO SEE THE TOP ELEMENT OF QUEUE
92. void peek(){
93.     if(front==-1 && rear==-1){
94.             printf("\n Queue is Empty\n");
95.     }
96.     else{
97.             printf("%d",queue[front]);
98.     }
99. }
100.
101.    //TO DISPLAY THE QUEUE
102.    void display(){
103.            int i;
104.            if(front==-1 && rear==-1){
105.                    printf("\nQueue is Empty\n");
106.            }
107.            else{
108.                    for(i=front;i<=rear;i++){
109.                            printf("%d ",queue[i]);
110.                    }
111.            }
112.    }
```


Implementation Of Deque

```
1.  /*
2.  implementation of deque
3.  */
4.
5.
6.  #include<stdio.h>
7.  #include<stdlib.h>
8.  #define N 5
9.
10. //Global Var Declarations
11. int F=-1,R=-1;
12. int dque[N];
13.
14. //Function declarations
15. void EnqueFront(int x);    //Insert From Front
16. void EnqueRear(int x);     //Insert From Rear
17. void DequeFront();         //Delete From Front
18. void DequeRear();          //Delete From Rear
19. void Display();            //Display
20.
```

```
21. //Driver Program
22. int main(){
23.       int choice,x;
24.       do{
25.             printf("\n\n**Deque Operactions**\n");
26.             printf("1. Enque Front\n");
27.             printf("2. Enque Rear\n");
28.             printf("3. Deque Front\n");
29.             printf("4. Deque Rear\n");
30.             printf("5. Display\n");
31.             printf("0 To exit\n");
32.             printf("Enter Choice");
33.             scanf("%d",&choice);
34.             switch(choice){
35.                   case 1:
36.                         printf("\nEnter Value ");
37.                         scanf("%d",&x);
38.                         EnqueFront(x);
39.                         break;
40.                   case 2:
41.                         printf("\nEnter Value ");
42.                         scanf("%d",&x);
43.                         EnqueRear(x);
44.                         break;
45.                   case 3:
46.                         DequeFront();
47.                         break;
48.                   case 4:
49.                         DequeRear();
50.                         break;
51.                   case 5:
52.                         Display();
53.                         break;
54.                   case 0:
55.                         exit(0);
56.                         break;
57.                   default:
58.                         printf("\n! Wrong Choice !");
59.             }
60.       }while(choice!=0);
61.       return 0;
62. }
63. //Insert From Front
64. void EnqueFront(int x){
65.       if(F==0 && R==N-1 || F==(R+1)){
66.             printf("\nQueue Is Full\n");
67.       }
68.       else if(F==-1 && R==-1){
69.             F=R=0;
70.             dque[F]=x;
71.       }
72.       else if(F==0){
73.             F=N-1;
74.             dque[F]=x;
75.       }
76.       else{
77.             F--;
78.             dque[F]=x;
79.       }
80. }
```

```
//Insert From Rear
void EnqueRear(int x){
	if(F==0 && R==N-1 || F==(R+1)){
		printf("\nQueue Is Full\n");
	}
	else if(F==-1 && R==-1){
		F=R=0;
		dque[R]=x;
	}
	else if(R==N-1){
		R=0;
		dque[R]=x;
	}
	else{
		R++;
		dque[R]=x;
	}
}

	//Delete From Front
	void DequeFront(){
		if(F==-1 && R==-1){
			printf("\nQueue Is Empty\n");
		}
		else if(F==R){
			F=R=-1;
		}
		else if(F==N-1){
			printf("%d ",dque[F]);
			F=0;
		}
		else{
			printf("%d ",dque[F]);
			F++;
		}
	}

	//Delete From Rear
	void DequeRear(){
		if(F==-1 && R==-1){
			printf("\nQueue Is Empty\n");
		}
		else if(F==R){
			F=R=-1;
		}
		else if(R==0){
			printf("%d",dque[R]);
			R=N-1;
		}
		else{
			printf("%d",dque[R]);
			R--;
		}
	}

	//To See the Deque
	void Display(){
		int i=F;
		if(F==-1 && R==-1){
			printf("\nQue Is Empty\n");
		}
		else{
			while(i!=R){
				printf("%d ",dque[i]);
				i=(i+1)%N;
			}
			printf(" %d",dque[i]);
		}
	   }
	}
```

Implementation Of Queue

```
/*
implementation Of Circular Queue
*/

#include<stdio.h>
#include<stdlib.h>
#define N 5

//GLOBAL
int queue[N];
int front=-1,rear=-1;

//Function Declarations
void Enque();
void Deque();
void Peek();
void Display();

//Driver Program
int main(){
	int choice;
	do{
		printf("\n***Circular Queue***\n");
		printf("1. Enque\n");
		printf("2. Deque\n");
		printf("3. Peek\n");
		printf("4. Display\n");
		printf("0 To Exit\n");
		printf("Enter Choice ");
		scanf("%d",&choice);
		switch(choice){
			case 1:
				Enque();
				break;
			case 2:
				Deque();
				break;
			case 3:
				Peek();
				break;
			case 4:
				Display();
				break;
			case 0:
				exit(0);
				break;
			default:
				printf("\n!Wrong Choice!\n");
		}
	}while(choice!=0);
	return 0;
}
```

```
//To insert element
void Enque(){
	int x;
	if(front==-1 && rear==-1){
		printf("\nEnter X ");
		scanf("%d",&x);
		front=rear=0;
		queue[rear]=x;
	}
	else if((rear+1)%N==front){
		printf("\nQueue Is Full\n");
	}
	else{
		printf("\nEnter X ");
		scanf("%d",&x);
		rear=(rear+1)%N;
		queue[rear]=x;
	}
}

//to delete element
void Deque(){
	if(front==-1 && rear==-1){
		printf("\nQueue Is Empty\n");
	}
	else if(front==rear){
		front=rear=-1;
	}
	else{
		printf("%d",queue[front]);
		front=(front+1)%N;
	}
}

//to see top of the queue
void Peek(){
	if(front==-1 && rear==-1){
		printf("\nQueue Is Empty\n");
	}
	else{
		printf("%d ",queue[front]);
	}
}

//to display circular queue
void Display(){
	int i=front;
		if(front==-1 && rear==-1){
			printf("\nQueue Is Empty\n");
		}
		else{
			while(i!=rear){
				printf("%d ",queue[i]);
				i=(i+1)%N;
			}
			printf("%d",queue[i]);
		}
	}
```