STACK :

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

int n,arr[5];
int Top=-1;
void Push();
void Pop();
void show();

void main()
{
   int choice;
   clrscr();
   printf("Enter size of an array:");
   scanf("%d",&n);
   while(1)
   {
        printf("\nOperations performed by Stack");
        printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");
        printf("\n\nEnter the choice:");
        scanf("%d",&choice);

        switch(choice)
        {
           case 1: Push();
                   break;
           case 2: Pop();
```

```
                    break;
            case 3: show();
                    break;
            case 0: exit(0);

            default: printf("\nInvalid choice!!");
        }
    }
    getch();
}


void Push()
{       int item;
        printf("value of n is : %d",n);
     if(Top==n-1)
    {
        printf("\nOverflow!!");
    }

    else

    {
        printf("\nEnter element to be inserted to the stack:");
        scanf("%d",&item);
        Top=Top+1;
        arr[Top]=item;
        printf("\n Element inserted successfully...\n");
    }
}
```

```
void Pop()
{
   if(Top==-1)
   {
        printf("\nUnderflow!!");
   }
   else
   {
        printf("\nPopped element:  %d",arr[Top]);
        Top=Top-1;
   }
}

void show()
{
   int i;

   if(Top==-1)
   {
        printf("\nUnderflow!!");
   }
   else
   {
        printf("\nElements present in the stack: \n");
        for(i=Top;i>=0;i--)
           printf("%d\n",arr[i]);
   }
}
```

SIMPLE QUEUE :

```
#include<conio.h>
#include<stdio.h>
int queue[5];
int i,n;
int front=-1,rear=-1;
void insert();
void deleted();
void display();
void main()
{
   int i, ch;
   printf("Enter the size:");
   scanf("%d",&n);
   do
   {
        printf("\nEnter your choice:\n 1.INSERT\n2.DELETE\n3.DISPLAY\n0.EXIT\n");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1:
                insert();
                break;
           case 2:
                deleted();
                break;
           case 3:
                display();
                break;
           default:
```

```
                printf("enter valid option...");
                break;
        }
    }while(ch!=0);
    getch();

}
void insert()
{
    int item;
    if(rear>=n-1)
    {
        printf("queue is overflow");
    }
    else
    {
        printf("Enter element to be inserted");
        scanf("%d",&item);
        rear=rear+1;
        queue[rear]=item;
        if(front==-1)
        {
            front=0;
        }
        printf("Insertion done... & item inserted=%d",item);
    }
}
void deleted()
{
    int item;
```

```
    if(front==-1)
    {
        printf("queue is underflow");
    }
    else
    {
        printf("Item deleted...%d",queue[front]);
        if(front==rear)
        {
            front=-1;
            rear=-1;
        }
        else
        {
            front=front+1;
        }
    }
}
void display()
{
    int i;
     if(front==-1)
    {
        printf("queue is empty");
    }
    else
    {
        for(i=front;i<=rear;i++)
        {
            printf("%d\n",queue[i]);
```

```
            }
        }
    }
```

CIRCULAR QUEUE

```
#include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
        if((front == 0 && rear == MAX-1) || (front == rear+1))
        {
                printf("Queue Overflow ");
        return;
        }

        if(front == -1)
        {
        front = 0;
        rear = 0;
        }
        else
        {
                if(rear == MAX-1)
                rear = 0;
                else
                rear = rear+1;
        }
        cqueue_arr[rear] = item ;
}
```

```
void deletion()
{
	if(front == -1)
	{
		printf("Queue Underflown");
		return ;
	}
	printf("Element deleted from queue is : %dn",cqueue_arr[front]);
	if(front == rear)
	{
		front = -1;
		rear=-1;
	}
	else
	{
		if(front == MAX-1)
		front = 0;
		else
		front = front+1;
	}
}
void display()
{
	int front_pos = front,rear_pos = rear;
	if(front == -1)
	{
		printf("Queue is emptyn");
		return;
	}
	printf("Queue elements :n");
```

```
        if( front_pos <= rear_pos )
        while(front_pos <= rear_pos)
        {
                printf("%d ",cqueue_arr[front_pos]);
                front_pos++;
        }
        else
        {
        while(front_pos <= MAX-1)
        {
                printf("%d ",cqueue_arr[front_pos]);
                front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
        {
                printf("%d ",cqueue_arr[front_pos]);
                front_pos++;
        }
        }
        printf("n");
}
int main()
{
        int choice,item;
        do
        {
        printf("1.Insertn");
        printf("2.Deleten");
        printf("3.Displayn");
```

```
        printf("4.Quitn");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1 :
        printf("Input the element for insertion in queue : ");
        scanf("%d", &item);
        insert(item);
        break;
        case 2 :
        deletion();
        break;
        case 3:
        display();
        break;
        case 4:
        break;
        default:
        printf("Wrong choicen");
        }
        }while(choice!=4);
return 0;
}
```

DQUEUE:

```
#include <stdio.h>
#include<conio.h>
#define size 5
int deque[size];
int f = -1, r = -1;
//  insert_front function will insert the value from the front
void insert_front(int x)
{
   if((f==0 && r==size-1))
   {
         printf("Overflow");
   }
   else if((f==-1) && (r==-1))
   {
         f=r=0;
         deque[f]=x;
   }
   else
   {
         f=f-1;
         deque[f]=x;
   }
}

// insert_rear function will insert the value from the rear
void insert_rear(int x)
{
   if((f==0 && r==size-1))
```

```
    {
        printf("Overflow");
    }
    else if((f==-1) && (r==-1))
    {
        r=0;
        deque[r]=x;
    }
    else
    {
        r++;
        deque[r]=x;
    }

}

// display function prints all the value of deque.
void display()
{
    int i=f;
    printf("\nElements in a deque are: ");

    while(i<=r)
    {
        printf("%d ",deque[i]);
        i++;
    }

}
```

```
// getfront function retrieves the first value of the deque.
void getfront()
{
   if((f==-1) && (r==-1))
   {
        printf("Deque is empty");
   }
   else
   {
        printf("\nThe value of the element at front is: %d", deque[f]);
   }

}

// getrear function retrieves the last value of the deque.
void getrear()
{
   if((f==-1) && (r==-1))
   {
        printf("Deque is empty");
   }
   else
   {
        printf("\nThe value of the element at rear is %d", deque[r]);
   }

}

// delete_front() function deletes the element from the front
void delete_front()
```

```
{
   if((f==-1) && (r==-1))
   {
         printf("Deque is empty");
   }
   else if(f==r)
   {
         printf("\nThe deleted element is %d", deque[f]);
         f=-1;
         r=-1;

   }

    else
    {
          printf("\nThe deleted element is %d", deque[f]);
          f=f+1;
    }
}

// delete_rear() function deletes the element from the rear
void delete_rear()
{
   if((f==-1) && (r==-1))
   {
         printf("Deque is empty");
   }
   else if(f==r)
   {
         printf("\nThe deleted element is %d", deque[r]);
```

```
        f=-1;
        r=-1;

    }

    else
    {
        printf("\nThe deleted element is %d", deque[r]);
        r=r-1;
    }
}

void main()
{
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);
    insert_rear(80);
    display();  // Calling the display function to retrieve the values of deque
    getfront();  // Retrieve the value at front-end
    getrear();  // Retrieve the value at rear-end
    delete_front();
    delete_rear();
    display(); // calling display function to retrieve values after deletion
    getch();
}
```

DOUBLY LINKEDLIST

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
void main ()
{
int choice =0;
   while(choice != 9)
   {
      printf("\n*********Main Menu*********\n");
      printf("\nChoose one option from the following list ...\n");
      printf("\n===============================================\n");
      printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete
from Beginning\n
      5.Delete from last\n6.Delete the node after the given
data\n7.Search\n8.Show\n9.Exit\n");
```

```
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
{
   case 1:
   insertion_beginning();
   break;
   case 2:
         insertion_last();
   break;
   case 3:
   insertion_specified();
   break;
   case 4:
   deletion_beginning();
   break;
   case 5:
   deletion_last();
   break;
   case 6:
   deletion_specified();
   break;
   case 7:
   search();
   break;
   case 8:
   display();
   break;
   case 9:
   exit(0);
```

```
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void insertion_beginning()
{
   struct node *ptr;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
    printf("\nEnter Item value");
    scanf("%d",&item);

   if(head==NULL)
   {
      ptr->next = NULL;
      ptr->prev=NULL;
      ptr->data=item;
      head=ptr;
   }
   else
   {
      ptr->data=item;
```

```
      ptr->prev=NULL;
      ptr->next = head;
      head->prev=ptr;
      head=ptr;
   }
   printf("\nNode inserted\n");
}

}
void insertion_last()
{
   struct node *ptr,*temp;
   int item;
   ptr = (struct node *) malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter value");
      scanf("%d",&item);
       ptr->data=item;
      if(head == NULL)
      {
         ptr->next = NULL;
         ptr->prev = NULL;
         head = ptr;
      }
      else
```

```
      {
        temp = head;
        while(temp->next!=NULL)
        {
           temp = temp->next;
        }
        temp->next = ptr;
        ptr ->prev=temp;
        ptr->next = NULL;
        }

      }
     printf("\nnode inserted\n");
    }
void insertion_specified()
{
   struct node *ptr,*temp;
   int item,loc,i;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\n OVERFLOW");
   }
   else
   {
      temp=head;
      printf("Enter the location");
      scanf("%d",&loc);
      for(i=0;i<loc;i++)
      {
```

```
            temp = temp->next;
            if(temp == NULL)
            {
               printf("\n There are less than %d elements", loc);
               return;
            }
         }
         printf("Enter value");
         scanf("%d",&item);
         ptr->data = item;
         ptr->next = temp->next;
         ptr -> prev = temp;
         temp->next = ptr;
         temp->next->prev=ptr;
         printf("\nnode inserted\n");
      }
   }
   void deletion_beginning()
   {
      struct node *ptr;
      if(head == NULL)
      {
         printf("\n UNDERFLOW");
      }
      else if(head->next == NULL)
      {
         head = NULL;
         free(head);
         printf("\nnode deleted\n");
      }
```

```
    else
    {
        ptr = head;
        head = head -> next;
        head -> prev = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }

}
void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
```

```
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void deletion_specified()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
    }
```

```
}
void display()
{
   struct node *ptr;
   printf("\n printing values...\n");
   ptr = head;
   while(ptr != NULL)
   {
      printf("%d\n",ptr->data);
      ptr=ptr->next;
   }
}
void search()
{
   struct node *ptr;
   int item,i=0,flag;
   ptr = head;
   if(ptr == NULL)
   {
      printf("\nEmpty List\n");
   }
   else
   {
      printf("\nEnter item which you want to search?\n");
      scanf("%d",&item);
      while (ptr!=NULL)
      {
         if(ptr->data == item)
         {
            printf("\nitem found at location %d ",i+1);
```

```
            flag=0;
            break;
        }
        else
        {
            flag=1;
        }
        i++;
        ptr = ptr -> next;
    }
    if(flag==1)
    {
        printf("\nItem not found\n");
    }
  }

}
```

SELECTION SORT

```
//selection sort

#include <stdio.h>
#include<conio.h>
void main()
{
int a[100], n, i, j, position, swap;
printf("Enter number of elementsn");
scanf("%d", &n);

printf("Enter %d Numbersn", n);
for (i = 0; i < n; i++)
scanf("%d", &a[i]);

        for(i = 0; i < n - 1; i++)
        {
                position=i;
                for(j = i + 1; j < n; j++)
                {
                if(a[position] > a[j])
                position=j;
                }

                if(position != i)
                {
                        swap=a[i];
                        a[i]=a[position];
                        a[position]=swap;
```

```
                }
        }
printf("Sorted Array:n");
for(i = 0; i < n; i++)
printf("%dn", a[i]);
getch();
}
```

LINEAR SEARCH

```
#include <stdio.h>
int linearSearch(int a[], int n, int val) {
  // Going through array sequencially
  for (int i = 0; i < n; i++)
    {
       if (a[i] == val)
       return i+1;
    }
  return -1;
}
int main() {
  int a[] = {70, 40, 30, 11, 57, 41, 25, 14, 52}; // given array
  int val = 41; // value to be searched
  int n = 9 // size of array
  int res = linearSearch(a, n, val); // Store result
  printf("The elements of the array are - ");
  for (int i = 0; i < n; i++)
  printf("%d ", a[i]);
  printf("\nElement to be searched is - %d", val);
  if (res == -1)
  printf("\nElement is not present in the array");
  else
  printf("\nElement is present at %d position of array", res);
  return 0;
}
```

BINARY SEARCH

```
// Binary Search in C

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {
  // Repeat until the pointers low and high meet each other
  while (low <= high) {
    int mid = low + (high - low) / 2;

    if (array[mid] == x)
      return mid;

    if (array[mid] < x)
      low = mid + 1;

    else
      high = mid - 1;
  }

  return -1;
}

int main(void) { 0                    6
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = 7;
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
```

```
  else
    printf("Element is found at index %d", result);
  return 0;
}
```

SINGLY LINKED LIST

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\n\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");
```

```
printf("\n=============================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n
5.Delete from last\n6.Delete node after specified location\n7.Search for an element\n8.Show\n9.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
{
   case 1:
   beginsert();
   break;
   case 2:
   lastinsert();
   break;
   case 3:
   randominsert();
   break;
   case 4:
   begin_delete();
   break;
   case 5:
   last_delete();
   break;
   case 6:
   random_delete();
   break;
   case 7:
   search();
   break;
   case 8:
```

```
            display();
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }

}
```

```
void lastinsert()
{
   struct node *ptr,*temp;
   int item;
   ptr = (struct node*)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter value?\n");
      scanf("%d",&item);
      ptr->data = item;
      if(head == NULL)
      {
         ptr -> next = NULL;
         head = ptr;
         printf("\nNode inserted");
      }
      else
      {
         temp = head;
         while (temp -> next != NULL)
         {
            temp = temp -> next;
         }
         temp->next = ptr;
         ptr->next = NULL;
         printf("\nNode inserted");
```

```
        }
    }
}
void randominsert()
{
    int i,loc,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }
```

```
        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
    }
}
void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }
}
void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
```

```
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}
void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);  2
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;
```

```
        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted node %d ",loc+1);
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
            }
```

```
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("Item not found\n");
        }
    }

}

void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values . . . . .\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
```

```
        }
    }
}
```

BUBBLE SORT

```
#include<stdio.h>
#include<conio.h>
int main(){
    int arr[50], num, x, y, temp;
    printf("Please Enter the Number of Elements you want in the array: ");
    scanf("%d\n", &num);
    printf("Please Enter the Value of Elements: ");
    for(x=0;x<num;x++)
        scanf("%d", &arr[x]);

    for(x=0;x<num-1;x++){
        for(y=0;y<num-x-1;y++){
            if(arr[y] > arr[y+1])
            {
                temp = arr[y];
                arr[y] = arr[y + 1];
                arr[y + 1] = temp;
            }
        }
    }
    printf("Array after implementing bubble sort: ");
    for(x=0;x<num;x++){
        printf("%d \n", arr[x]);
```

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