//Display twin prime numbers from 1 to 100

```
#include<stdio.h>
int prime(int);
int main()
{
    int i,n=0;
    printf("The twin primes between 1 to 100 :\n");
    for(i=3;i<98;i=i+2)
    {
        if(prime(i)&&prime(i+2))
        {
            printf("{%d,%d}\n",i,i+2);
            n++;
        }
    }
    return 0;
}
int prime(int a)

{
    int j;
    for(j=2;j<=a/2;j++)
        if((a%j)==0)
            break;
    if(j>a/2)
        return(1);
    return(0);
}
```

//Linear search

```
#include <stdio.h>
int main()
{
  int arr[100], x, c, n;

  printf("Enter number of elements in array: ");
  scanf("%d", &n);

  printf("Enter %d integers:", n);

  for (c = 0; c < n; c++)
    scanf("%d", &arr[c]);

  printf("Enter a number to search:");
  scanf("%d", &x);
  for (c = 0; c < n; c++)
  {
    if (arr[c] == x)
    {
      printf("%d is present at index %d.\n", x, c);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", x);

  return 0;
}
```

//String reversal

```
#include<stdio.h>
void main()
{
    int i, j, k;
    char str[100];
    char rev[100];
    printf("Enter a string:");
    scanf("%s", str);
    printf("The original string is %s\n", str);
    for(i = 0; str[i] != '\0'; i++);
    {
        k = i-1;
    }
    for(j = 0; j <= i-1; j++)
    {
        rev[j] = str[k];
        k--;
    }
    printf("The reverse string is %s\n", rev);
    getch();
}
```

//Create initialise and use pointers

```
#include<stdio.h>
int main(){
       int n=5;
       int*j;
       j=&n;
       printf("Value of n: %d\n",*j);
       printf("Address of n: %u",j);
       return 0;
}
```

//Add two numbers using pointers

```
#include<stdio.h>
int main(){
	int n1,n2,sum;
	int*i,*j;
	i=&n1;
	j=&n2;
	printf("Enter the first number: ");
	scanf("%d",i);
	printf("Enter the second number: ");
	scanf("%d",j);
	sum=*i+*j;
	printf("Sum of two numbers: %d",sum);
	return 0;
}
```

//Swap two numbers using pointers.

```
#include<stdio.h>
int main(){
	int n1,n2,temp;
	int*i,*j;
	i=&n1;
	j=&n2;
	printf("Before swapping\n");
	printf("Enter the first element:");
	scanf("%d",i);
	printf("Enter the second element:");
	scanf("%d",j);
	temp = *i;
	*i = *j;
	*j = temp;
	printf("After swapping\n");
	printf("First element:%d\n",*i);
	printf("Second element:%d",*j);
	return 0;
}
```

//Input and print array elements using pointers.

```
#include<stdio.h>
int main(){
	int a[100],n,i;
	int *p = a;
	printf("Enter the size of array:");
	scanf("%d",&n);
	printf("Enter the array elements:\n");
	for(i=0;i<n;++i){
		scanf("%d",(p+i));
	}
	printf("Array elements are:\n");
	for(i=0;i<n;++i){
		printf("%d\n",*(p+i));
	}
	return 0;
}
```

//Sum of the series 1!/1+2!/2+3!/3+4!/4+5!/5 using functions

```
#include<stdio.h>
int fact();
int main(){
	int sum;
	sum = fact(1)/1 + fact(2)/2 + fact(3)/3 + fact(4)/4 + fact(5)/5;
	printf("Sum of the series is: %d",sum);
	return 0;
}
int fact(int n){
	int num=0,f=1;
	while(num<=n-1){
		f=f+f*num;
		num++;
	}
	return f;
}
```

//Array

```
#include <stdio.h>
int main()
{
	int arr[100] = { 0 };
	int i, x, pos, n, a, del, ser;
	printf("Enter number of elements in array: ");
	scanf("%d", &n);
	printf("Enter %d integers:", n);
	for (i = 0; i < n; i++)
		scanf("%d", &arr[i]);
	printf("Options are: \n");
	printf("1-insertion \n");
	printf("2-deletion \n");
	printf("3-searching \n");
	printf("4-display elements \n");
	printf("\n");
	printf("Please select the options:");
	scanf("%d",&a);
	if(a==1){
		printf("Enter the element to be inserted:");
		scanf("%d",&x);
		printf("Enter the position:");
		scanf("%d",&pos);
		n++;
	for (i = n; i >= pos; i--)
		arr[i] = arr[i - 1];
		arr[pos] = x;
	for (i = 0; i < n; i++)
		printf("%d ", arr[i]);
		printf("\n");
	}
	if(a==2){
		printf("Enter the location to delete element:");
		scanf("%d", &del);
			if (del >= n+1)
			printf("Deletion not possible.\n");
			 else
				{
				for (i = del; i < n - 1; i++)
				arr[i] = arr[i+1];
				printf("Resultant array:\n");
				for (i = 0; i < n - 1; i++)
			printf("%d\n", arr[i]);
				}
	}
	if(a==3){
		printf("Enter a number to search:");
		scanf("%d", &ser);
		for (i = 0; i < n; i++)
```

```
        {
        if (arr[i] == ser)
    {
        printf("%d is present at index %d.\n", ser, i);
        break;
    }
        }
               if (i == n)
        printf("%d isn't present in the array.\n", ser);
        }
        if(a==4){
               for (i = 0; i < n; i++)
               printf("%d ", arr[i]);
        }
        return 0;
}
```

//Selection sort

```
#include<stdio.h>
int main(){
   int i, j, count, temp, number[100];
   printf("Enter number of elements: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&number[i]);
   for(i=0;i<count;i++){
      for(j=i+1;j<count;j++){
         if(number[i]>number[j]){
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }
   }
   printf("Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);

   return 0;
}
```

//Bubble sort

```
#include<stdio.h>
int main(){
   int count, temp, i, j, number[100];
   printf("Enter number of elements: ");
   scanf("%d",&count);
   printf("Enter %d numbers: ",count);
   for(i=0;i<count;i++)
   scanf("%d",&number[i]);
   for(i=count-2;i>=0;i--){
      for(j=0;j<=i;j++){
        if(number[j]>number[j+1]){
           temp=number[j];
           number[j]=number[j+1];
           number[j+1]=temp;
        }
      }
   }
   printf("Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);
   return 0;
}
```

//Insertion sort

```
#include<stdio.h>
int main(){
   int i, j, count, temp, number[100];
   printf("Enter number of elements: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&number[i]);
   for(i=1;i<count;i++){
      temp=number[i];
      j=i-1;
      while((temp<number[j])&&(j>=0)){
         number[j+1]=number[j];
         j=j-1;
      }
      number[j+1]=temp;
   }
   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);
   return 0;
```

```
}
```

//Quick sort

```
#include<stdio.h>
void quicksort(int num[25],int a,int b){
   int i, j, pivot, temp;

   if(a<b){
      pivot=a;
      i=a;
      j=b;

      while(i<j){
         while(num[i]<=num[pivot]&&i<b)
            i++;
         while(num[j]>num[pivot])
            j--;
         if(i<j){
            temp=num[i];
            num[i]=num[j];
            num[j]=temp;
         }
      }

      temp=num[pivot];
      num[pivot]=num[j];
      num[j]=temp;
      quicksort(num,a,j-1);
      quicksort(num,j+1,b);

   }
}

int main(){
   int i, count, num[25];

   printf("Enter the number of elements: ");
   scanf("%d",&count);

   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&num[i]);

   quicksort(num,0,count-1);

   printf(" Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",num[i]);

   return 0;
}
```

//Merge sort

```
#include <stdio.h>
#define MAX 30

int main()
{
	int arr[MAX],temp[MAX],i,j,k,n,size,l1,h1,l2,h2;
	printf("Enter the number of elements : ");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter element %d : ",i+1);
		scanf("%d",&arr[i]);
	}
	printf("Unsorted list is : ");
	for( i = 0 ; i<n ; i++)
		printf("%d ", arr[i]);
		printf("\n");
	for(size=1; size < n; size=size*2 )
	{
		l1=0;
		k=0;
		while( l1+size < n)
		{
			h1=l1+size-1;
			l2=h1+1;
			h2=l2+size-1;
			if( h2>=n )
				h2=n-1;
			i=l1;
			j=l2;
			while(i<=h1 && j<=h2 )
			{
				if( arr[i] <= arr[j] )
					temp[k++]=arr[i++];
				else
					temp[k++]=arr[j++];
			}
			while(i<=h1)
				temp[k++]=arr[i++];
			while(j<=h2)
				temp[k++]=arr[j++];

			l1=h2+1;
		}
		for(i=l1; k<n; i++)
			temp[k++]=arr[i];
```

```
                for(i=0;i<n;i++)
                        arr[i]=temp[i];
        }
        printf("Sorted list is : ");
        for( i = 0 ; i<n ; i++)
                printf("%d ", arr[i]);
        return 0;
}
```

// Singly linked list

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

struct linked_list
{
    int number;
    struct linked_list *next;
};

typedef struct linked_list node;
node *head=NULL, *last=NULL;

void create_linked_list();
void print_linked_list();
void insert_at_last(int value);
void insert_at_first(int value);
void insert_after(int key, int value);
void delete_item(int value);
void search_item(int value);

int main()
{
    int key, value,options;

    printf("Create Linked List\n");
    create_linked_list();
    print_linked_list();
    printf("1-Insert value at last position\n");
    printf("2-Insert value at first position \n");
    printf("3-Insert value after a defined value\n");
    printf("4-Search an item from Linked List\n");
    printf("5-Delete value from List\n");
    printf("6-Quit the program\n");
       printf("Select the options:");
    scanf("%d",&options);


       if(options==1) {

       printf("\nInsert new item at last:\n");
           scanf("%d", &value);
           insert_at_last(value);
           print_linked_list();
}
```

```
    if(options==2){
        printf("Insert new item at first:\n");
           scanf("%d", &value);
           insert_at_first(value);
           print_linked_list();

       }



    if(options==3){
       printf("\nEnter a element (existing item of List), after that
you want to insert a value:\n");
           scanf("%d", &key);
           printf("\nInsert new item after KEY:\n");
           scanf("%d", &value);
           insert_after(key, value);
           print_linked_list();

       }


    if(options==4){
        printf("\nEnter an item to search it from List:\n");
           scanf("%d", &value);
           search_item(value);

       }


   if(options==5){
       printf("\nEnter a value which you want to delete from list:\n");
           scanf("%d", &value);
           delete_item(value);
           print_linked_list();
       }
       if(options==6){
              printf("Quit");
       }


    return 0;
}


void create_linked_list()
{
    int val;
    while(1)
    {
        printf("Input a number. (Enter -1 to exit)\n");
       scanf("%d", &val);
        if(val==-1)
            break;
        insert_at_last(val);
```

```
    }
}

void insert_at_last(int value)
{
    node *temp_node;
    temp_node = (node *) malloc(sizeof(node));
    temp_node->number=value;
    temp_node->next=NULL;


    if(head==NULL)
    {
        head=temp_node;
        last=temp_node;
    }
    else
    {
        last->next=temp_node;
        last=temp_node;
    }

}


void insert_at_first(int value)
{
    node *temp_node = (node *) malloc(sizeof(node));
    temp_node->number=value;
    temp_node->next = head;
    head = temp_node;
}

void insert_after(int key, int value)
{
    node *myNode = head;
    int flag = 0;
    while(myNode!=NULL)
    {
        if(myNode->number==key)
        {
            node *newNode = (node *) malloc(sizeof(node));
            newNode->number = value;
            newNode->next = myNode->next;
            myNode->next = newNode;
            printf("%d is inserted after %d\n", value, key);
            flag = 1;
            break;
        }
        else
            myNode = myNode->next;
    }
    if(flag==0)
        printf("Key not found!\n");
}

void delete_item(int value)
{
```

```
    node *myNode = head, *previous=NULL;
    int flag = 0;
    while(myNode!=NULL)
    {
        if(myNode->number==value)
        {
            if(previous==NULL)
                head = myNode->next;
            else
                previous->next = myNode->next;
            printf("%d is deleted from list\n", value);
            flag = 1;
            free(myNode);
            break;
        }
        previous = myNode;
        myNode = myNode->next;
    }

    if(flag==0)
        printf("Key not found!\n");
}
void search_item(int value)
{
    node *searchNode = head;
    int flag = 0;
    while(searchNode!=NULL)
    {
        if(searchNode->number==value)
        {
            printf("%d is present in this list. Memory address is
%d\n", value, searchNode);
            flag = 1;
            break;
        }
        else
            searchNode = searchNode->next;
    }
    if(flag==0)
        printf("Item not found\n");
}
void print_linked_list()
{
    printf("\nYour full linked list is\n");
       node *myList;
    myList = head;
    while(myList!=NULL)
    {
        printf("%d ", myList->number);
        myList = myList->next;
    }
    puts("");
}
```

//Doubly Linked List

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

struct node
{
    struct node *prev;
    int n;
    struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();

int count = 0;

void main()
{
    int ch;

    h = NULL;
    temp = temp1 = NULL;

    printf("\n 1 - Insert at beginning");
    printf("\n 2 - Insert at end");
    printf("\n 3 - Insert at position i");
    printf("\n 4 - Delete at i");
    printf("\n 5 - Display from beginning");
    printf("\n 6 - Display from end");
    printf("\n 7 - Search for element");
    printf("\n 8 - Sort the list");
    printf("\n 9 - Update an element");
    printf("\n 10 - Exit");
```

```
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert1();
            break;
        case 2:
            insert2();
            break;
        case 3:
            insert3();
            break;
        case 4:
            delete();
            break;
        case 5:
            traversebeg();
            break;
        case 6:
            temp2 = h;
            if (temp2 == NULL)
                printf("\n Error : List empty to display ");
            else
            {
                printf("\n Reverse order of linked list is : ");
                traverseend(temp2->n);
            }
            break;
        case 7:
            search();
            break;
        case 8:
            sort();
            break;
        case 9:
            update();
            break;
        case 10:
            exit(0);
        default:
            printf("\n Wrong choice menu");
        }
    }
}

/* TO create an empty node */
void create()
{
    int data;

    temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node : ");
    scanf("%d", &data);
```

```
    temp->n = data;
    count++;
}

/*  TO insert at beginning */
void insert1()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}

/* To insert at end */
void insert2()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp1->next = temp;
        temp->prev = temp1;
        temp1 = temp;
    }
}

/* To insert at any position */
void insert3()
{
    int pos, i = 2;

    printf("\n Enter position to be inserted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Position out of range to insert");
        return;
    }
    if ((h == NULL) && (pos != 1))
    {
        printf("\n Empty list cannot insert other than 1st position");
        return;
```

```
    }
    if ((h == NULL) && (pos == 1))
    {
        create();
        h = temp;
        temp1 = h;
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        create();
        temp->prev = temp2;
        temp->next = temp2->next;
        temp2->next->prev = temp;
        temp2->next = temp;
    }
}

/* To delete an element */
void delete()
{
    int i = 1, pos;

    printf("\n Enter position to be deleted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Error : Position out of range to delete");
        return;
    }
    if (h == NULL)
    {
        printf("\n Error : Empty list no elements to delete");
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        if (i == 1)
        {
            if (temp2->next == NULL)
            {
                printf("Node deleted from list");
                free(temp2);
                temp2 = h = NULL;
                return;
            }
```

```
        }
        if (temp2->next == NULL)
        {
            temp2->prev->next = NULL;
            free(temp2);
            printf("Node deleted from list");
            return;
        }
        temp2->next->prev = temp2->prev;
        if (i != 1)
            temp2->prev->next = temp2->next;    /* Might not need this
statement if i == 1 check */
        if (i == 1)
            h = temp2->next;
        printf("\n Node deleted");
        free(temp2);
    }
    count--;
}

/* Traverse from beginning */
void traversebeg()
{
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("List empty to display \n");
        return;
    }
    printf("\n Linked list elements from begining : ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}

/* To traverse from end recursively */
void traverseend(int i)
{
    if (temp2 != NULL)
    {
        i = temp2->n;
        temp2 = temp2->next;
        traverseend(i);
        printf(" %d ", i);
    }
}

/* To search for an element in the list */
void search()
{
    int data, count = 0;
    temp2 = h;
```

```
    if (temp2 == NULL)
    {
        printf("\n Error : List empty to search for data");
        return;
    }
    printf("\n Enter value to search : ");
    scanf("%d", &data);
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
            printf("\n Data found in %d position",count + 1);
            return;
        }
        else
             temp2 = temp2->next;
            count++;
    }
    printf("\n Error : %d not found in list", data);
}

/* To update a node value in the list */
void update()
{
    int data, data1;

    printf("\n Enter node data to be updated : ");
    scanf("%d", &data);
    printf("\n Enter new data : ");
    scanf("%d", &data1);
    temp2 = h;
    if (temp2 == NULL)
    {
        printf("\n Error : List empty no node to update");
        return;
    }
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {

            temp2->n = data1;
            traversebeg();
            return;
        }
        else
            temp2 = temp2->next;
    }

    printf("\n Error : %d not found in list to update", data);
}

/* To sort the linked list */
void sort()
{
    int i, j, x;

    temp2 = h;
```

```
    temp4 = h;

    if (temp2 == NULL)
    {
        printf("\n List empty to sort");
        return;
    }

    for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
    {
        for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
        {
            if (temp2->n > temp4->n)
            {
                x = temp2->n;
                temp2->n = temp4->n;
                temp4->n = x;
            }
        }
    }
    traversebeg();
}
```

//Circular linked list

```
#include <stdio.h>
#include <stdlib.h>
/*Structure of node with data and a pointer */
struct node {
int data;
struct node * next;
}*head;

void createList(int n);
void displayList();
void insert_beginning(int data);
void search_element(int data);
void updating_element(int data);
void insert_given_position(int data, int position);
void delete_beginning();
void delete_given_position();
void reverse_list();

int main()
{
int n, data, choice=1;
head = NULL;
while(choice != 0)
{
printf("\n\n\t\tCIRCULAR LINKED LIST BASIC OPERATIONS\n\n");
printf("1. Create List\n");
printf("2. Insert at beginning\n");
printf("3. Insert at any position\n");
printf("4. Delete at beginning\n");
printf("5. Delete at any position\n");
printf("6. Search Element\n");
printf("7. Update Element\n");
printf("8. Reverse List\n");
printf("0. Exit\n");
printf("\n\n");
printf("Enter your choice : ");
scanf("%d", &choice);
switch(choice)
{
case 1:
printf("Enter the total number of nodes in list: ");
scanf("%d", &n);
createList(n);
displayList();
break;
case 2:
printf("Enter data to be inserted at beginning: ");
scanf("%d", &data);
insert_beginning(data);
displayList();
break;
case 3:
printf("Enter node position: ");
```

```
scanf("%d", &n);
printf("Enter data you want to insert at %d position: ", n);
scanf("%d", &data);
insert_given_position(data, n);
displayList();
break;
case 4:
if(head == NULL)
{
printf("\nThe list is empty\n");
}
else
{
delete_beginning();
displayList();
}
break;
case 5:
if(head == NULL)
{
printf("\nThe list is empty\n");
}
else
{
delete_given_position();
displayList();
}
break;
case 6:
printf("\nEnter the element to be searched : ");
scanf("%d",&data);
search_element(data);
break;
case 7:
printf("\nEnter the element to be updated : ");
scanf("%d",&data);
updating_element(data);
printf("\nThe Updated List is\n\n");
displayList();
break;
case 8:
reverse_list();
break;
case 0:
break;
default:
printf("Error! Invalid choice.");
}
printf("\n");
}
return 0;
}

void createList(int n)
{
int i, data;
struct node *prevNode, *newNode;
if(n >= 1)
```

```
{
//Creates and links the head node
head = (struct node *)malloc(sizeof(struct node));
printf("Data of node 1 : ");
scanf("%d", &data);
head->data = data;
head->next = NULL;
prevNode = head;

//Creates and links rest of the n-1 nodes
for(i=2; i<=n; i++)
{
newNode = (struct node *)malloc(sizeof(struct node));
printf("Data of node %d : ", i);
scanf("%d", &data);
newNode->data = data;
newNode->next = NULL;

prevNode->next = newNode; //Links the previous node with newly created
node
prevNode = newNode; //Moves the previous node ahead
}

prevNode->next = head; //Links the last node with first node
}
}

void displayList()
{
struct node *current;
int n = 1;
if(head == NULL)
{
printf("List is empty.\n");
}
else
{
current = head;
printf("The SINGLY CIRCULAR LINKED LIST IS : \n");
do {
printf("%d\t",current->data);
current = current->next;
n++;
}while(current != head);
}
}

void insert_beginning(int data)
{
struct node *newNode, *current;
if(head == NULL)
{
printf("List is empty.\n");
}
else
{
//Creates new node, assign data and links it to head
newNode = (struct node *)malloc(sizeof(struct node));
```

```
newNode->data = data;
newNode->next = head;
printf("\nThe element %d is inserted at the beginning",data);
printf("\n");

//Traverses to last node and links last node with first node which is
new node
current = head;
while(current->next != head)
{
current = current->next;
}
current->next = newNode;

head = newNode; //Makes new node as head node
}
}

void insert_given_position(int data, int position)
{
struct node *newNode, *current;
int i;
if(head == NULL)
{
printf("List is empty.\n");
}
else if(position == 0)
{
insert_beginning(data);
}
else
{
//Creates new node and assign data to it
newNode = (struct node *)malloc(sizeof(struct node));
newNode->data = data;
printf("\nThe element %d is inserted at index %d",data,position);
printf("\n");

//Traverse to n-1 node
current = head;
for(i=2; i<=position; i++)
{
current = current->next;
}
//Links new node with node ahead of it and previous to it
newNode->next = current->next;
current->next = newNode;
}
}

void delete_beginning()
{
struct node * temp,*s;
// When start and end of the list are same
if (head == head->next)
{
head = NULL;
printf("\nThe List is empty\n");
```

```
}
else
{
temp = head;
s = head;
while (temp->next != head) // Traverse until start node is reached
again
{
temp = temp -> next;
}
printf("\nThe element %d is deleted at the beginning",s -> data);
printf("\n");
head = s->next; // shift head node
temp->next = head;
printf("\n");
free(s);
}
}

void delete_given_position()
{
struct node * temp, *s;
if (head == NULL)
printf("\nThe List is empty");
else
{
int count = 0, pos;
printf("\nEnter the position to be deleted : ");
scanf("%d", &pos);
temp = head;
while (count < pos)
{
s = temp;
temp = temp -> next;
count++;
}
printf("\nThe element %d at index %d is deleted",temp -> data,pos);
printf("\n");
s -> next = temp -> next;
printf("\n");
free(temp);
}
}

void search_element(int data)
{
struct node * temp = head;
int index = 0;
while(temp)
{
if(temp -> data == data)
{
printf("\nElement found at index %d in the list",index); //Print if the
element is found
break;
}
else
{
```

```
temp = temp -> next;
index++;
}
}
}

void updating_element(int data)
{
int new_data;
printf("\nEnter the new data to replace with : ");
scanf("%d",&new_data);
struct node * temp = head;
while(temp)
{
if(temp -> data == data)
{
temp -> data = new_data; //Update the element if found and exit the
loop
break;
}
else
{
temp = temp -> next;
}
}
}

void reverse_list()
{
struct node *prev, *cur, *next, *last;
printf("\nThe reversed List is\n\n");
// Cannot reverse empty list
if (head == NULL)
{
printf("Cannot reverse empty list.\n");
return;
}

// Head is going to be our last node after reversing list
last = head;
prev = head;
cur = (head)->next;
head = (head)->next;
// Iterate till you reach the initial node in circular list
while (head != last)
{
head = head -> next;
cur->next = prev;
prev = cur;
cur = head;
}
cur->next = prev;
head = prev;
displayList();
}
```

//Stack using array

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int arr[MAX];
int top = -1;

void push(int item){
	if(top == MAX-1)
	{
		printf("Stack overflow.");
	}
	else{
		top = top+1;
		arr[top] = item;
	}
}
int pop(){
	int item;
	if(top == -1){
		printf("Stack underflow.");
	}
	else{
		item = arr[top] ;
		top = top-1;
		return item;
	}
}
int peek(){
	if(top == -1){
		printf("Stack underflow.");
	}
	else{
		return arr[top];
	}
}
void display(){
	int i;
	if(top == -1){
		printf("Stack underflow.");
	}
	else{
		printf("Stack elements are:");
		for(i=top;i>=0;i--)
			printf("%d\n",arr[i]);
		printf("\n");

	}
}
int main(){
	int choice,item;
	while(1){
```

```
                printf("\nOptions are 1)push 2)pop 3)peek 4)display
5)Quit\n");
                printf("Enter the options:");
                scanf("%d",&choice);
                switch(choice){
                        case 1:
                                printf("Enter the element to be
inserted:");
                                scanf("%d",&item);
                                push(item);
                                break;
                        case 2:
                                item = pop();
                                printf("The element to be popped is :
%d\n",item);
                                break;
                        case 3:
                                printf("The element at the top is
%d\n",peek());
                                break;
                        case 4:
                                display();
                                break;
                        case 5:
                                exit(1);
                        default:
                                printf("Wrong choice");
                }

        }

return 0;
}
```

//Stack using linked list

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

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


void push ()
{
    int data;
    struct node *ptr;
       ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
```

```
    {
        printf("not able to push the element");
    }
    else
    {
        printf("Enter the value: ");
        scanf("%d",&data);
        if(head==NULL)
        {
            ptr->data = data;
            ptr -> next = NULL;
            head=ptr;
        }
        else
        {
            ptr->data = data;
            ptr->next = head;
            head=ptr;

        }
        printf("Item pushed");

    }
}

void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow");
    }
    else
    {
        item = head->data;
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped");

    }
}
void display()
{
    int i;
    struct node *ptr;
    ptr=head;
    if(ptr == NULL)
    {
        printf("Stack is empty\n");
    }
    else
    {
        printf("Printing Stack elements: \n");
        while(ptr!=NULL)
        {
            printf("%d\n",ptr->data);
```

```
            ptr = ptr->next;
        }
    }
}
void main ()
{
    int choice=0;
    while(1)
    {
        printf("\nChoose one from the below options:\n");
        printf("1.Push 2.Pop 3.Display 4.Exit\n");
        printf("\n Enter your choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
               exit(1)  ;
            }
            default:
            {
                printf("Please Enter valid choice ");
            }
    };
}
}
```

//Queue using array

```
#include <stdio.h>

#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
```

```
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n",
queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /* End of display() */
```

//Queue using linked list

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

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

void insert(int);
void delete();
void display();

void main()
{
   int choice, value;
```

```
   printf("\n:: Queue Implementation using Linked List ::\n");
   {
      printf("\n*** MENU ***\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      while(choice!=4){
       if (choice == 1)
       {
         printf("Enter the value to be insert: ");
               scanf("%d", &value);
               insert(value);}

       else if (choice == 2)
         delete();
        else if (choice == 3)
         display();
        printf("Enter your choice: ");
      scanf("%d",&choice);
      }
   }
}
void insert(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode -> next = NULL;
   if(front == NULL)
      front = rear = newNode;
   else{
      rear -> next = newNode;
      rear = newNode;
   }
}
void delete()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      front = front -> next;
      printf("\nDeleted element: %d\n", temp->data);
      free(temp);
   }
}
void display()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
      }
      printf("%d--->NULL\n",temp->data);
   }}
```

//Double ended queue

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

int deque[MAX];
int left=-1, right=-1;

void insert_right();
void insert_left();
void delete_right();
void delete_left();
void display();

int main()
{
 int choice;
 do
 {
  printf("\n1.Insert at right ");
  printf("\n2.Insert at left ");
  printf("\n3.Delete from right ");
  printf("\n4.Delete from left ");
  printf("\n5.Display ");
  printf("\n6.Exit");
  printf("\n\nEnter your choice ");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1:
    insert_right();
    break;
   case 2:
    insert_left();
    break;
   case 3:
    delete_right();
    break;
   case 4:
    delete_left();
    break;
   case 5:
    display();
    break;
  }
 }while(choice!=6);
```

```
 getch();
 return 0;
}

void insert_right()
{
 int val;
 printf("\nEnter the value to be added ");
 scanf("%d",&val);
 if( (left==0 && right==MAX-1) || (left==right+1) )
 {
  printf("\nOVERFLOW");
 }
 if(left==-1)
 {
  left=0;
  right=0;
 }
 else
 {
  if(right==MAX-1)
   right=0;
  else
   right=right+1;
 }
 deque[right]=val;
}


void insert_left()
{
 int val;
 printf("\nEnter the value to be added ");
 scanf("%d",&val);
 if( (left==0 && right==MAX-1) || (left==right+1) )
 {
  printf("\nOVERFLOW");
 }
 if(left==-1)
 {
  left=0;
  right=0;
 }
 else
 {
  if(left==0)
   left=MAX-1;
  else
   left=left-1;
 }
 deque[left]=val;
}



void delete_right()
{
 if(left==-1)
```

```
 {
  printf("\nUNDERFLOW");
  return;
 }
 printf("\nThe deleted element is %d\n", deque[right]);
 if(left==right)
 {
  left=-1;
  right=-1;
 }
 else
 {
  if(right==0)
   right=MAX-1;
  else
   right=right-1;
 }
}



void delete_left()
{
 if(left==-1)
 {
  printf("\nUNDERFLOW");
  return;
 }
 printf("\nThe deleted element is %d\n", deque[left]);
 if(left==right)
 {
  left=-1;
  right=-1;
 }
 else
 {
  if(left==MAX-1)
   left=0;
  else
   left=left+1;
 }
}


void display()
{
 int front=left, rear=right;
 if(front==-1)
 {
  printf("\nQueue is Empty\n");
  return;
 }
 printf("\nThe elements in the queue are: ");
 if(front<=rear)
 {
  while(front<=rear)
  {
   printf("%d\t",deque[front]);
```

```
   front++;
  }
 }
 else
 {
  while(front<=MAX-1)
  {
   printf("%d\t",deque[front]);
   front++;
  }
  front=0;
  while(front<=rear)
  {
   printf("%d\t",deque[front]);
   front++;
  }
 }
 printf("\n");
}
```

//Binary search tree

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

struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

void delete1();
void insert();
void delete();
void inorder(struct btnode *t);
void create();
void search(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
void search1(struct btnode *t,int data);
int smallest(struct btnode *t);
int largest(struct btnode *t);

int flag = 1;

void main()
{
    int ch;

    printf("\nOPERATIONS ---");
```

```
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Delete an element from the tree\n");
    printf("3 - Inorder Traversal\n");
    printf("4 - Preorder Traversal\n");
    printf("5 - Postorder Traversal\n");
    printf("6 - Exit\n");
    while(1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            preorder(root);
            break;
        case 5:
            postorder(root);
            break;
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* To insert a node in the tree */
void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

/* To create a node */
void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}
```

```
/* Function to search the appropriate position to insert the new node
*/
void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))      /* value more
than root node value insert at right */
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))    /* value
less than root node value insert at left */
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

/* recursive function to perform inorder traversal of tree */
void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d -> ", t->value);
    if (t->r != NULL)
        inorder(t->r);
}

/* To check for the deleted node */
void delete()
{
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

/* To find the preorder traversal */
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
```

```
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

/* To find the postorder traversal */
void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}

/* Search for the appropriate position to insert the new node */
void search1(struct btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}

/* To delete a node */
void delete1(struct btnode *t)
{
    int k;

    /* To delete leaf node */
    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
```

```
}

/* To delete node having one left hand child */
else if ((t->r == NULL))
{
    if (t1 == t)
    {
        root = t->l;
        t1 = root;
    }
    else if (t1->l == t)
    {
        t1->l = t->l;

    }
    else
    {
        t1->r = t->l;
    }
    t = NULL;
    free(t);
    return;
}

/* To delete node having right hand child */
else if (t->l == NULL)
{
    if (t1 == t)
    {
        root = t->r;
        t1 = root;
    }
    else if (t1->r == t)
        t1->r = t->r;
    else
        t1->l = t->r;
    t == NULL;
    free(t);
    return;
}

/* To delete node having two child */
else if ((t->l != NULL) && (t->r != NULL))
{
    t2 = root;
    if (t->r != NULL)
    {
        k = smallest(t->r);
        flag = 1;
    }
    else
    {
        k =largest(t->l);
        flag = 2;
    }
    search1(root, k);
    t->value = k;
}
```

```
}

/* To find the smallest element in the right sub tree */
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}

/* To find the largest element in the left sub tree */
int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```

//Recursive tree traversal

```
#include <stdio.h>

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

struct node* newNode(int item)
{
    struct node* temporary =  (struct node *)malloc(sizeof(struct
node));
    temporary->data = item;
    temporary->left = temporary->right = NULL;
    return temporary;
}

void postorder(struct node* root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
```

```
        printf("%d ", root->data);
    }
}

void preorder(struct node* root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(struct node* root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}


struct node* insert(struct node* node, int data)
{

    if (node == NULL) return newNode(data);


    if (data < node->data)
        node->left  = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);

    return node;
}

int main()
{
    struct node* root = NULL;
    root = insert(root, 9);
    insert(root, 7);
    insert(root, 5);
    insert(root, 8);
    insert(root, 14);
    insert(root, 11);
    insert(root, 16);

    int choice;
    printf("1. Postorder 2. PreOrder 3. Inorder\nEnter your
choice:\n");
    scanf("%d",&choice);

    switch(choice){
        case 1:
        printf("The postorder is :\n");
```

```
        postorder(root);
        break;

        case 2:
        printf("The preorder is :\n");
        preorder(root);
        break;

        case 3:
        printf("The inorder is :\n");
        inorder(root);
break;


        default:
        printf("Enter correct choice: \n");
    }

    return 0;
}
```

//Non recursive tree traversal

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 50

struct node
{
        struct node *lchild;
        int info;
        struct node *rchild;
};

struct node *insert_nrec(struct node *root, int ikey );
void nrec_pre(struct node *root);
void nrec_in(struct node *root);
void nrec_post(struct node *root);
void display(struct node *ptr,int level);

struct node *queue[MAX];
int front=-1,rear=-1;
void insert_queue(struct node *item);
struct node *del_queue();
int queue_empty();

struct node *stack[MAX];
int top=-1;
void push_stack(struct node *item);
struct node *pop_stack();
int stack_empty();
```

```
int main( )
{
        struct node *root=NULL, *ptr;
        int choice,k;

        while(1)
        {
                printf("\n");
                printf("1.Insert\n");
                printf("2.Display\n");
                printf("3.Preorder Traversal\n");
                printf("4.Inorder Traversal\n");
                printf("5.Postorder Traversal\n");
                printf("6.Quit\n");
                printf("\nEnter your choice : ");
                scanf("%d",&choice);

                switch(choice)
                {

                case 1:
                        printf("\nEnter the key to be inserted : ");
                        scanf("%d",&k);
                        root = insert_nrec(root, k);
                        break;

        case 2:
                        printf("\n");
                        display(root,0);
                        printf("\n");
                        break;

                case 3:
                        nrec_pre(root);
                        break;

                case 4:
                        nrec_in(root);
                        break;

                case 5:
                        nrec_post(root);
                        break;

                case 6:
                        exit(1);

                default:
                        printf("\nWrong choice\n");
                }
        }

        return 0;

}
```

```
struct node *insert_nrec(struct node *root, int ikey)
{
        struct node *tmp,*par,*ptr;

        ptr = root;
        par = NULL;

        while( ptr!=NULL)
        {
                par = ptr;
                if(ikey < ptr->info)
                        ptr = ptr->lchild;
                else if( ikey > ptr->info )
                        ptr = ptr->rchild;
                else
                {
                        printf("\nDuplicate key");
                        return root;
                }
        }

        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=ikey;
        tmp->lchild=NULL;
        tmp->rchild=NULL;

        if(par==NULL)
                root=tmp;
        else if( ikey < par->info )
                par->lchild=tmp;
        else
                par->rchild=tmp;

        return root;
}


void nrec_pre(struct node *root)
{
        struct node *ptr = root;
        if( ptr==NULL )
        {
                printf("Tree is empty\n");
                return;
        }
        push_stack(ptr);
        while( !stack_empty() )
        {
                ptr = pop_stack();
                printf("%d  ",ptr->info);
                if(ptr->rchild!=NULL)
                        push_stack(ptr->rchild);
                if(ptr->lchild!=NULL)
                        push_stack(ptr->lchild);
        }
        printf("\n");
}
```

```
void nrec_in(struct node *root)
{
        struct node *ptr=root;

        if( ptr==NULL )
        {
                printf("Tree is empty\n");
                return;
        }
        while(1)
        {
      while(ptr->lchild!=NULL )
                {
                        push_stack(ptr);
                        ptr = ptr->lchild;
                }

                while( ptr->rchild==NULL )
                {
                        printf("%d  ",ptr->info);
                        if(stack_empty())
                                return;
                        ptr = pop_stack();
                }
                printf("%d  ",ptr->info);
                ptr = ptr->rchild;
        }
        printf("\n");
}

void nrec_post(struct node *root)
{
        struct node *ptr = root;
        struct node *q;

        if( ptr==NULL )
        {
                printf("Tree is empty\n");
                return;
        }
        q = root;
        while(1)
        {
                while(ptr->lchild!=NULL)
                {
                        push_stack(ptr);
                        ptr=ptr->lchild;
                }

                while( ptr->rchild==NULL || ptr->rchild==q )
                {
                        printf("%d  ",ptr->info);
                        q = ptr;
                        if( stack_empty() )
                                return;
                        ptr = pop_stack();
                }
                push_stack(ptr);
```

```
                ptr = ptr->rchild;
        }
        printf("\n");
}
void insert_queue(struct node *item)
{
        if(rear==MAX-1)
        {
                printf("Queue Overflow\n");
                return;
        }
        if(front==-1)
                front=0;
        rear=rear+1;
        queue[rear]=item ;
}

struct node *del_queue()
{
        struct node *item;
        if(front==-1 || front==rear+1)
        {
                printf("Queue Underflow\n");
                return 0;
        }
        item=queue[front];
        front=front+1;
        return item;
}

int queue_empty()
{
        if(front==-1 || front==rear+1)
                return 1;
        else
                return 0;
}


void push_stack(struct node *item)
{
        if(top==(MAX-1))
        {
                printf("Stack Overflow\n");
                return;
        }
        top=top+1;
        stack[top]=item;
}

struct node *pop_stack()
{
        struct node *item;
        if(top==-1)
        {
                printf("Stack Underflow....\n");
                exit(1);
        }
```

```
        item=stack[top];
        top=top-1;
        return item;
}

int stack_empty()
{
        if(top==-1)
                return 1;
        else
                return 0;
}


void display(struct node *ptr,int level)
{
        int i;
        if(ptr == NULL )
                return;
        else
    {
                display(ptr->rchild, level+1);
                printf("\n");
                for (i=0; i<level; i++)
                        printf("    ");
                printf("%d", ptr->info);
                display(ptr->lchild, level+1);
        }
}
```

//DFS

```
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;    //n is no of vertices and graph is
sorted in array G[10][10]

void main()
{
    int i,j;
    printf("Enter number of vertices:");

        scanf("%d",&n);

    //read the adjecency matrix
        printf("\nEnter adjecency matrix of the graph:");

        for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                        scanf("%d",&G[i][j]);

    //visited is initialized to zero
```

```
   for(i=0;i<n;i++)
        visited[i]=0;

    DFS(0);
}

void DFS(int i)
{
    int j;
       printf("\n%d",i);
    visited[i]=1;

       for(j=0;j<n;j++)
       if(!visited[j]&&G[i][j]==1)
            DFS(j);}
```

//BFS

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

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
       create_graph();
       BF_Traversal();
       return 0;
}

void BF_Traversal()
{
       int v;

       for(v=0; v<n; v++)
              state[v] = initial;

       printf("Enter Start Vertex for BFS: \n");
       scanf("%d", &v);
```

```
        BFS(v);
}

void BFS(int v)
{
        int i;

        insert_queue(v);
        state[v] = waiting;

        while(!isEmpty_queue())
        {
                v = delete_queue( );
                printf("%d ",v);
                state[v] = visited;

                for(i=0; i<n; i++)
                {
                        if(adj[v][i] == 1 && state[i] == initial)
                        {
                                insert_queue(i);
                                state[i] = waiting;
                        }
                }
        }
        printf("\n");
}

void insert_queue(int vertex)
{
        if(rear == MAX-1)
                printf("Queue Overflow\n");
        else
        {
                if(front == -1)
                        front = 0;
                rear = rear+1;
                queue[rear] = vertex ;
        }
}

int isEmpty_queue()
{
        if(front == -1 || front > rear)
                return 1;
        else
                return 0;
}

int delete_queue()
{
        int delete_item;
        if(front == -1 || front > rear)
        {
                printf("Queue Underflow\n");
                exit(1);
        }
```

```
        delete_item = queue[front];
        front = front+1;
        return delete_item;
}

void create_graph()
{
        int count,max_edge,origin,destin;

        printf("Enter number of vertices : ");
        scanf("%d",&n);
        max_edge = n*(n-1);

        for(count=1; count<=max_edge; count++)
        {
                printf("Enter edge %d( -1 -1 to quit ) : ",count);
                scanf("%d %d",&origin,&destin);

                if((origin == -1) && (destin == -1))
                        break;

                if(origin>=n || destin>=n || origin<0 || destin<0)
                {
                        printf("Invalid edge!\n");
                        count--;
                }
                else
                {
                        adj[origin][destin] = 1;
                }
        }
}
```

//Dijisktra's algorithm

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
        int G[MAX][MAX],i,j,n,u;
        printf("Enter no. of vertices:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");

        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&G[i][j]);
```

```
        printf("\nEnter the starting node:");
        scanf("%d",&u);
        dijkstra(G,n,u);

        return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

        int cost[MAX][MAX],distance[MAX],pred[MAX];
        int visited[MAX],count,mindistance,nextnode,i,j;


        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        if(G[i][j]==0)
                                cost[i][j]=INFINITY;
                        else
                                cost[i][j]=G[i][j];

        for(i=0;i<n;i++)
        {
                distance[i]=cost[startnode][i];
                pred[i]=startnode;
                visited[i]=0;
        }

        distance[startnode]=0;
        visited[startnode]=1;
        count=1;

        while(count<n-1)
        {
                mindistance=INFINITY;


                for(i=0;i<n;i++)
                        if(distance[i]<mindistance&&!visited[i])
                        {
                                mindistance=distance[i];
                                nextnode=i;
                        }


                        visited[nextnode]=1;
                        for(i=0;i<n;i++)
                                if(!visited[i])

        if(mindistance+cost[nextnode][i]<distance[i])
                                        {

        distance[i]=mindistance+cost[nextnode][i];
                                                pred[i]=nextnode;
                                        }
                count++;
        }
```

```
        for(i=0;i<n;i++)
                if(i!=startnode)
                {
                        printf("\nDistance of node%d=%d",i,distance[i]);
                        printf("\nPath=%d",i);

                        j=i;
                        do
                        {
                                j=pred[j];
                                printf("<-%d",j);
                        }while(j!=startnode);
        }
}
```