

LOURDES MATHA

COLLEGE OF SCIENCE AND TECHNOLOGY

Bureau Veritas Certified ISO 9001-2015 Institution

Managed by Archdiocese of Changanacherry



DATA STRUCTURES LAB(20MCA135)

Name :

University Reg No:LMC21MCA-.....

Class: ...MCA.....Semester.....S1.....

Year:2021-2023.....

From Page No:1.....To.....88.....

Certified Bonafide Record of Work Done By

.....

External Examiner

Head of the Department

Faculty-in-Charge

Place: Kuttichal

Prof. SELMA JOSEPH

Prof. SHERIN JOSEPH

Date:

INDEX

SL.NO	NAME OF THE EXPERIMENT	DATE	PAGE.NO
1	Merge two sorted array and store in third array	01/11/2021	14
2	Stack using Linked List	08/11/2021	18
3	Circular Queue	17/11/2021	25
4	Doubly Linked List	24/11/2021	31
5	Binary Search Tree Traversal and Deletion	01/12/2021	43
6	Set Operation Using Disjoint Set	06/12/2021	56
7	Set Operation Using Bit String	13/12/2021	62
8	Bredth First Search	03/01/2022	69
9	Depth First Search	10/01/2022	73
10	Construct MST using Prim's Algorithm	17/01/2022	79
11	Topological Sorting	26/01/2022	88

INTRODUCTION TO DATA STRUCTURE

Data structure is a representation of data and the operations allowed on that data. A data structure is a way to store and organize data in order to facilitate the access and modifications. Data structure is representation of the logical relationship existing between individual elements of data. A data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other. Data structure affects the design of both structural & functional aspects of a program.

Program=Algorithm + Data Structure

Algorithm:- A set of instruction written to carry out certain tasks.

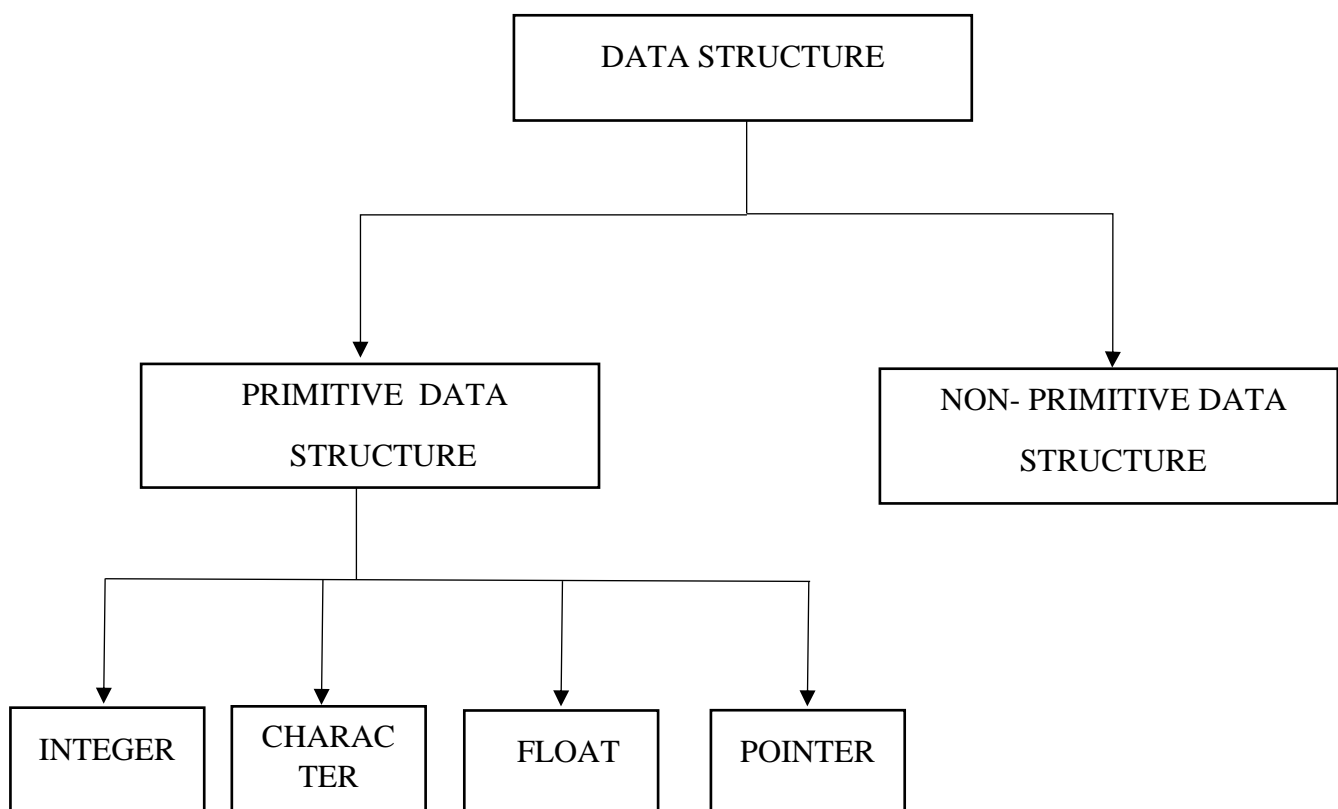
Data structure :-The way of organizing the data with their logical relationship retained.

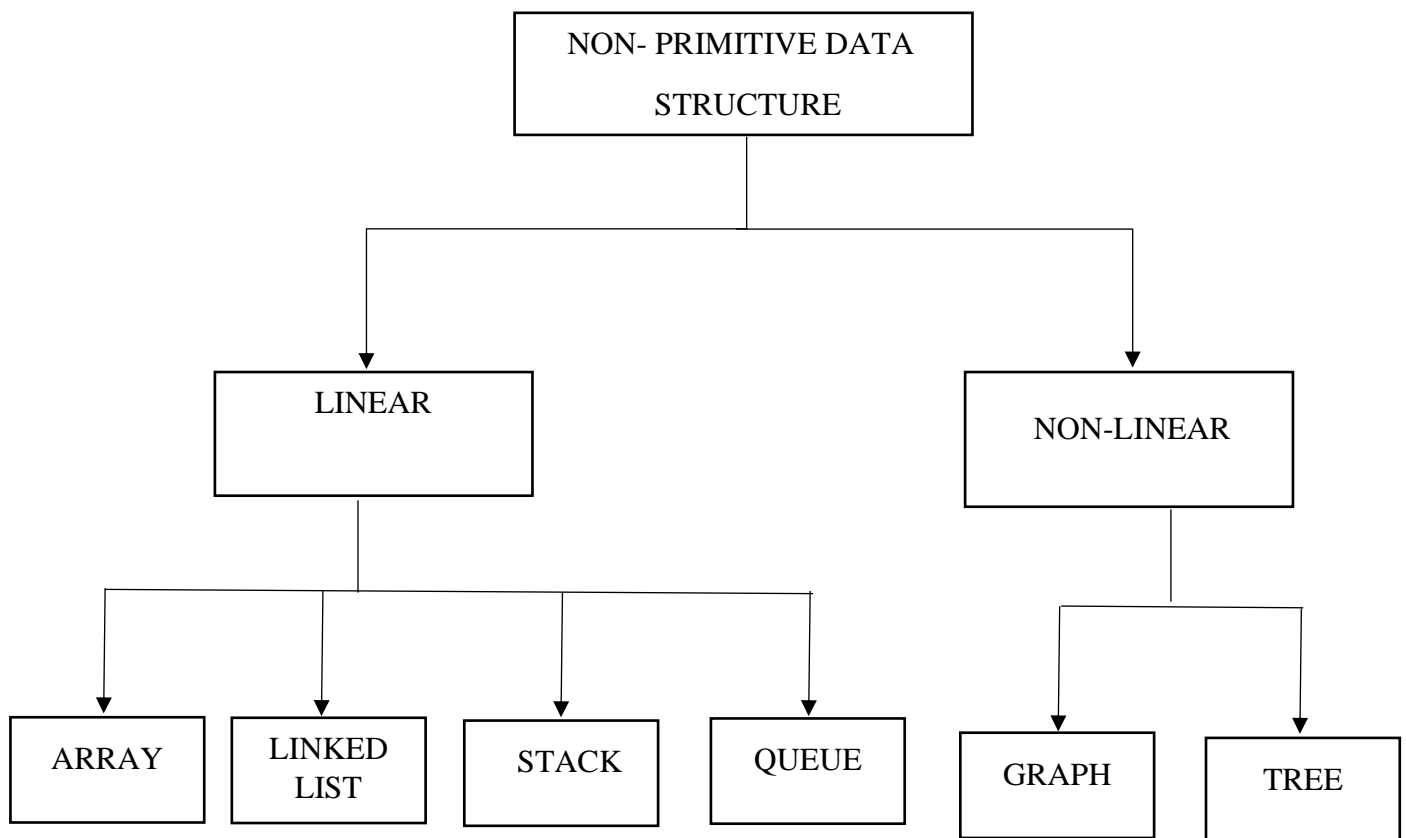
To develop a program of an algorithm, we should select an appropriate data structure for that algorithm. Therefore algorithm and its associated data structures form a program.

CLASSIFICATION OF DATA STRUCTURE

Data structure are normally divided into two broad categories:

- **Primitive Data Structure**
- **Non-Primitive Data Structure**





PRIMITIVE DATA STRUCTURE

- Basic structures and directly operated upon by the machine instructions.
- There are different representation on different computers.
- Integer, Floating-point number, Character constants, pointers etc.

NON-PRIMITIVE DATA STRUCTURE

- More sophisticated data structures.
- Derived from the primitive data structures.
- The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.
- Lists, Stack, Queue, Tree, Graph are example of non-primitive data structures.
- The design of an efficient data structure must take operations to be performed on the data structure.

STACK DATA STRUCTURE

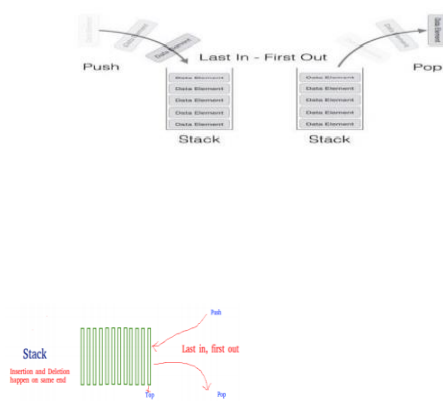
Stack is a linear data structure which follows a particular order in which the operations are performed. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc. A real-world stack allows operations at one end only.



For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. The element which is placed (inserted or added) last, is accessed first.

Stack Representation

The following diagram depicts a stack and its operations



Stack can either be a fixed size one or it may have a sense of dynamic resizing. We are implementing stack using arrays, which makes it a fixed size stack implementation.

Basic Operations

A stack is used for the following two primary operations

PUSH()

Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition – In stack terminology, insertion operation is called PUSH operation.

POP()

Removing (accessing) an element from the stack. In stack terminology removal operation is called POP operation. When data is PUSH ed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks

- peek() – get the top data element of the stack, without removing it.
- isFull() – check if stack is full.
- isEmpty() – check if stack is empty.
- Push: add an item to the stack.
- Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- Peek or Top: Returns top element of stack.
- isEmpty: Returns true if stack is empty, else false.

At all times, we maintain a pointer to the last PUSH ed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides top value of the stack without actually removing it.

Push Operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps

- Step 1 – Checks if the stack is full.
- Step 2 – If the stack is full, produces an error and exit.
- Step 3 – If the stack is not full, increments top to point next empty space.
- Step 4 – Adds data element to the stack location, where top is pointing.
- Step 5 – Returns success.

Pop Operation

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

A Pop operation may involve the following steps

- Step 1 – Checks if the stack is empty.
- Step 2 – If the stack is empty, produces an error and exit.
- Step 3 – If the stack is not empty, accesses the data element at which top is pointing.
- Step 4 – Decreases the value of top by 1.
- Step 5 – Returns success.

QUEUE DATA STRUCTURE

Queue is a linear data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology (FIFO), i.e., the data item stored first will be accessed first.

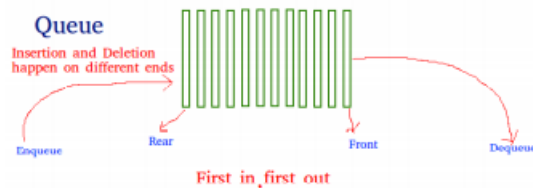
A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first.



More real-world examples can be seen as queues at the ticket windows and bus-stops.

Queue Representation

The following diagram given below tries to explain queue representation as data structure



As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures. For the sake of simplicity, we shall implement queues using one-dimensional array.

Basic Operations

Basic operations associated with queues are

- enqueue() – add (store) an item to the queue.
- dequeue() – remove (access) an item from the queue.

Enqueue Operation

Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks. The following steps should be taken to enqueue (insert) data into a queue

Step 1 – Check if the queue is full.

Step 2 – If the queue is full, produce overflow error and exit.

Step 3 – If the queue is not full, increment rear pointer to point the next empty space.

Step 4 – Add data element to the queue location, where the rear is pointing.

Step 5 – return success.

Dequeue Operation

Accessing data from the queue is a process of two tasks I. Access the data where front is pointing II. Removes the data after access. The following steps are taken to perform dequeue operation

Step 1 – Check if the queue is empty.

Step 2 – If the queue is empty, produce underflow error and exit.

Step 3 – If the queue is not empty, access the data where front is pointing.

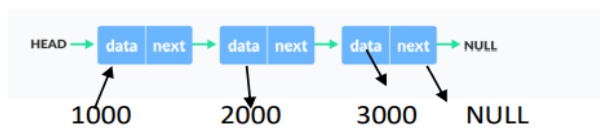
Step 4 – Increment front pointer to point to the next available data element.

Step 5 – Return success.

LINKED LIST

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In simple words, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list. A linked list data structure includes a series of connected nodes. You have to start somewhere, so we give the address of the first node a special name called HEAD. Also, the last node in the linked list can be identified because its next portion points to NULL.

For example,



Representation of Linked List

Each node consists:

- I. A data item
- II. An address of another node

We wrap both the data item and the next node reference in a struct as:

struct node

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

Each struct node has a data item and a pointer to another struct node.

TYPES OF LINKED LIST

There are three common types of Linked List.

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List

- It is the most common.
- Each node has data and a pointer to the next node.



Doubly Linked List

- We add a pointer to the previous node in a doubly-linked list.
- Thus, we can go in either direction: forward or backward.



Circular Linked List

- A circular linked list is a variation of a linked list in which the last element is linked to the first element.
- This forms a circular loop.



A circular linked list can be either singly linked or doubly linked.

- For singly linked list, next pointer of last item points to the first item
- In the doubly linked list, prev pointer of the first item points to the last item as well.

TREE

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.

Tree Vocabulary:

The topmost node is called root of the tree. The elements that are directly under an element are called its children. The element directly above something is called its parent. For example, 'a' is a child of 'f', and 'f' is the parent of 'a'. Finally, elements with no children are called leaves.

```
tree
----
j  <-- root
/  \
f    k
/  \  \
a   h   z  <-- leaves
```

USE OF TREE

1. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:
file system

```
-----
/  <-- root
/  \
...  home
    /  \
    ugrad  course
    /  /  |  \
...  cs101 cs112 cs113
```

2. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).

3. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).

4. Like Linked Lists and unlike Arrays, Trees don't have an upper limit on number of nodes as nodes are linked using pointers.

Main applications of trees include:

1. Manipulate hierarchical data.

2. Make information easy to search (see tree traversal).

3. Manipulate sorted lists of data.

4. As a workflow for compositing digital images for visual effects.

5. Router algorithms

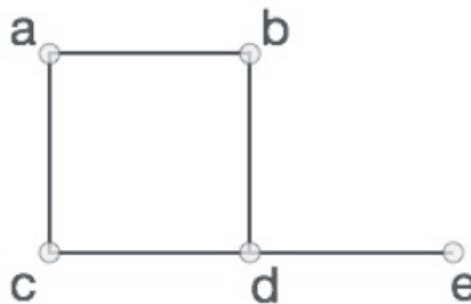
6. Form of a multi-stage decision-making (see business chess).

GRAPH

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Formally, a graph is a pair of sets (**V**, **E**), where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph



In the above graph,

$V = \{a, b, c, d, e\}$

$E = \{ab, ac, bd, cd, de\}$

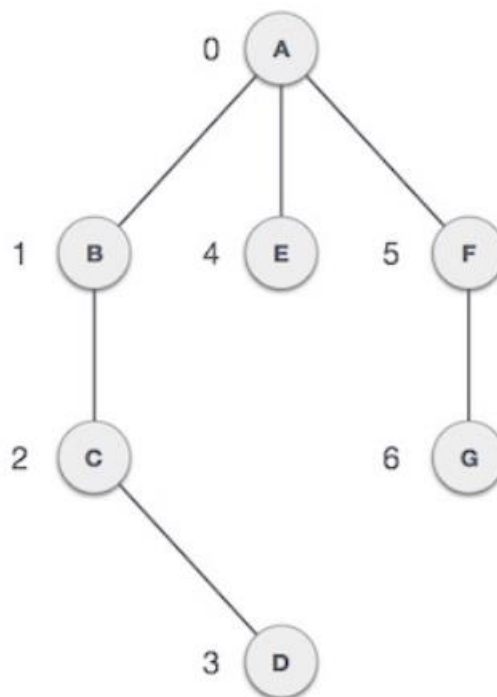
Graph Data Structure

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms

- **Vertex** – Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
- **Edge** – Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image.

Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.

- **Adjacency** – Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
- **Path** – Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



PROGRAM NUMBER :01

Date:01/11/2021

AIM: Write a C program merge two sorted array and stored in third array

ALGORITHM

Step 1:start

Step 2:read the size of array1

Step 3:enter the elements of array1

Step 4:read the size of second array

Step 5:enter the elements of array2

Step 6:initialize i=0,j=0

Step 7:while(i<m&& j<n)

Step 8:if(arr1[i]<arr[j]) then

Step 9: array3[k] = array1[i] and increment i

Step 10:else

Step 11: array3[k] = array1[j] and increment j

Step 12:increment k

Step 13:if(i>=m) then

Step 14:while(j<n)

Step 15: array3[k] = array1[j] and increment j&k

Step 16:if(j>=n) then

Step 17: array3[k] = array1[i] and increment i&k

Step 18:print the array after merging

Step 19:stop

PROGRAM

```
#include <stdio.h>

void main()
{
    int array1[50], array2[50], array3[100], m, n, i, j, k = 0;
    printf("\n Enter size of array Array 1: ");
    scanf("%d", &m);
    printf("\n Enter sorted elements of array 1: \n");
    for (i = 0; i < m; i++)
    {
        scanf("%d", &array1[i]);
    }
    printf("\n Enter size of array 2: ");
    scanf("%d", &n);
    printf("\n Enter sorted elements of array 2: \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &array2[i]);
    }
    i = 0;
    j = 0;
    while (i < m && j < n)
    {
        if (array1[i] < array2[j])
        {
            array3[k] = array1[i];
            i++;
        }
        else
        {

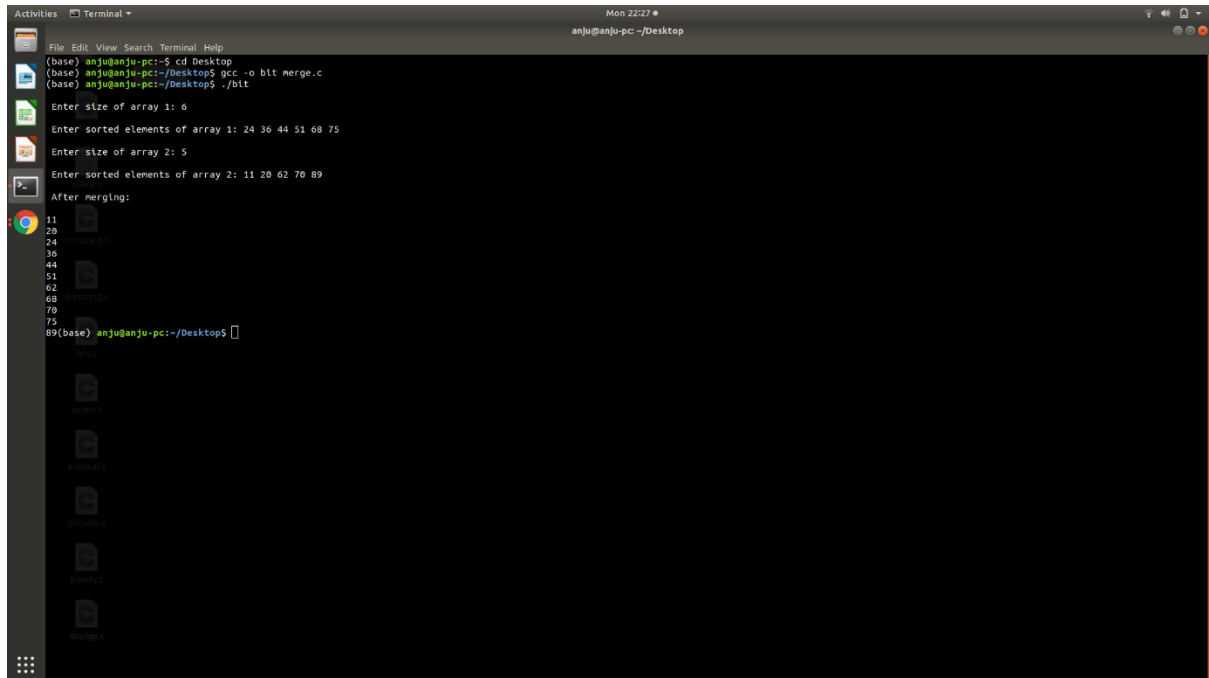
```

```

        array3[k] = array2[j];
        j++;
    }
    k++;
}
if (i >= m)
{
    while (j < n)
    {
        array3[k] = array2[j];
        j++;
        k++;
    }
}
if (j >= n)
{
    while (i < m)
    {
        array3[k] = array1[i];
        i++;
        k++;
    }
}
printf("\n After merging: \n");
for (i = 0; i < m + n; i++)
{
    printf("\n%d", array3[i]);
}
}

```


OUTPUT



```
Activities Terminal Mon 22:27
anju@anju-pc: ~/Desktop
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit merge.c
(base) anju@anju-pc:~/Desktop$ ./bit
Enter size of array 1: 6
Enter sorted elements of array 1: 24 36 44 51 68 75
Enter size of array 2: 5
Enter sorted elements of array 2: 11 20 62 70 89
After merging:
11
20
24
36
44
51
62
68
70
75
89
anju@anju-pc:~/Desktop$
```

AIM: Write a C program to implement stack using linked list

ALGORITHM

Step 1:start

Step 2:Define a structure 'node' with two member val and next

Step 3:Define a node pointer head

Step 4:enter the choice

Step 5:if choice==1 then do push operation

Step 6: if choice==2 then do pop operation

Step 7:if choice==3 then display linked list

Step 8:if choice==4 then exit

Step 9:stop

Push operation

Step 1:start

Step 2:create a new node

Step 3:if(head==NULL) then

Step 4:ptr->val=val

Step 5:ptr->next=NULL

Step 6:head=ptr

Step 7:else

Step 8:ptr->val = val

Step 9:ptr->next = head

Step 10:head=ptr

Step 11:print item pushed

Step 12:stop

Pop operation

Step 1:start

Step 2:if(head==NULL) then

Step 3:print underflow

Step 4:else

Step 5:item = head->val

Step 6:ptr = hea

Step 7:head = head->next

Step 8:free(ptr)

Step 9:print Item popped

Step 10:stop

Display linked list

Step 1:start

Step 2:if (ptr==NULL) then

Step 3:print stack is empty

Step 4:else

Step 5:while(ptr!=NULL)

Step 6:print the elments in the stack

Step 7:stop

PROGRAM

```
#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node *next;

};

struct node *head;

void main ()

{

int choice=0;

printf("\n*****Stack operations using linked list*****\n");

printf("\n-----\n");

while(choice != 4)

{

printf("\n\nChose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

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:
        {
            printf("Exiting....");
            break;
        }
        default:
        {
            printf("Please Enter valid choice ");
        }
    };
}
}

void push ()
{
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("not able to push the element");
    }
}

```

```

else
{
    printf("Enter the value");
    scanf("%d",&val);
    if(head==NULL)
    {
        ptr->val = val;
        ptr -> next = NULL;
        head=ptr;
    }
    else
    {
        ptr->val = val;
        ptr->next = head;
        head=ptr;
    }
    printf("Item pushed");
}
}

void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow");
    }
    else
    {
        item = head->val;

```

```

    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->val);
            ptr = ptr->next;
        }
    }
}

```

OUTPUT

Activities Terminal File Edit View Search Terminal Help Sat 21:06 anju@anju-pc: ~/Desktop

```
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit stack.c
(base) anju@anju-pc:~/Desktop$ ./bit

*****Stack operations using linked list*****
-----

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 1
Enter the value: 1
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 1
Enter the value: 2
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 1
Enter the value: 3
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 1
Enter the value: 4
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
```

```

Activities Terminal Sat 21:07 anju@anju-pc ~/Desktop
File Edit View Search Terminal Help
4.Exit
Enter your choice: 1
Enter the value: 3
Item pushed
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 1
Enter the value: 4
Item pushed
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 1
Enter the value: 5
Item pushed
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 2
Item popped
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 3
Printing Stack elements
4
3
2
1
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
Enter your choice: 

```


AIM:Write a C program to implement circular queue

ALGORITHM

step 1:start
step 2:enter the size of circular queue and assign to n
step 3:enter the choice
step 4:if choice=1 then do insertion
step 5:if choice=2 then do deletion
step 6:if choice=3 then display circular queue
step 8:if choice=4 then exit
step 9:stop

Insert operation

step 1:start
step 2: if ((front== 1 && rear == n) || (front == rear+ 1)) then print queue is full
step 3:elseif(front == NULL) then front =rear=1
step 4:else if (rear == n) then rear = 1
step 5:else rear++
step 6:read the element to inserted and assign to y
step 7:queue[rear]=y
step 8:stop

Delete operation

step 1:start
step 2:if(front== NULL) then print queue is empty
step 3:else
step 4:delete queue[front]
step 5:if(front==rear) then front=rear=NULL
step 6:else if(front==n) then front=1

step 7:else front ++

step 8:stop

Display circular queue

step 1:start

step 2:if (front>rear) then

step 3:for (i = front; i <=n; i++)

step 4: print queue[i])

step 5:for (i = 1; i <= rear; i++)

step 6:print queue[i]

step 7:else

step 8:for (i = front; i <= rear; i++)

step 9:print queue[i]

step 10:stop

PROGRAM

```
#include<stdio.h>

int queue[30],n,front=0,rear=0,c,y;

void insert();

void delete();

void display();

void main()

{

printf("Enter the size of circular queue\n");

scanf("%d",&n);

do

{

printf("\t\t\nMENU\t\t\n");

printf("\n1. Insert \n2. Delete\n3. Display\n4. Exit");

printf("\n\nEnter Choice 1-4? : ");

scanf("%d", &c);

switch (c)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:printf("exit\n");

}

}while (c!= 4);
```

```

}

void insert()
{
    if ((front== 1 && rear == n) || (front == rear+ 1))
    {
        printf("queue is full");
        return;
    }
    else
    { if (front == NULL)
        front =rear=1;
        else if (rear == n)
            rear = 1;
        else
            rear++;
        printf("Enter the element to be inserted\n");
        scanf("%d",&y);
    }
    queue[rear]=y;
}

void delete()
{
    if (front == NULL)
    {
        printf("Queue is empty ");
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        if (front == rear)

```

```

        front = rear=NULL;
    else
        if(front==n)
            front=1;
        else
            front++;
    }
}
void display()
{
    int i;
    if (front > rear)
    {
        printf("The Circular Queue is");
        for (i = front; i <=n; i++)
            printf("%d\t ", queue[i]);
        for (i = 1; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    else
    {
        for (i = front; i <= rear; i++)
            printf("%d\t ", queue[i]);
    }
}

```

OUTPUT

```
Activities Terminal Sat 22:21
anju@anju-pc: ~/Desktop
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit circularq.c
(base) anju@anju-pc:~/Desktop$ ./bit

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 12

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 26

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 38

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 50

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 73

Circular Queue:
```

```
Activities Terminal Sat 22:20
anju@anju-pc: ~/Desktop
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 50

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 1
Enter number: 73

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 2
12 deleted

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 2
26 deleted

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 3
38
50
73

Circular Queue:
0. Exit
1. Insert
2. Delete
3. Display
Enter Choice 0-3: 
```

PROGRAM NUMBER :04

Date:24/11/2021

AIM:Write a C program to implement Doubly linked list

ALGORITHM

step 1:start

step 2:define a structure node

step 3:initialize data and pointers *previous, *next and assign *head=NULL

step 4:read the choice

step 5:if choice=1 then do insert at a position

step 6:read the value to inserted and assign to value

step 7:select the insertion position

step 8:if ch=1 then insert at beginning

step 9:if ch=2 then insert at end

step 10:if ch=3 then insert after a node

step 11:if choice=2 then do select the deleting position

step 12:if ch=1 then delete at beginning

step 13:if ch=2 then delete at end

step 14:if ch=2 then delete a specific node

step 15:if choice=3 then do

step 16: if(head == NULL) then print List is Empty

step 17: else initialize *temp=head

step 18: while(temp -> next != NULL)

step 19:print temp->data

step 20:if choice=4 then exit

step 21:stop

Insert at beginning

step 1:start
step 2:create a newNode
step 3:newNode->data=value
step 4:newNode->previous=NULL
step 5:if (head==NULL) then
step 6: newNode->next=NULL
step 7:head=newNode
step 8:else newNode->next=NULL
step 9:head=newNode
step 10:print insertion success
step 11:stop

Insert at end

step 1:start
step 2:create a newNode
step 3:newNode->data=value
step 4:newNode->next=NULL
step 5:if(head==NULL) then
step 6:newNode -> previous = NULL
step 7:head = newNode
step 8:else
step 9:*temp=head
step 10:while(temp->next!=NULL)
step 11:temp=temp -> next
step 12:temp -> next = newNode
step 13:newNode -> previous = temp
step 14:print insertion success
step 15:stop

Insert after a node

step 1:start
step 2:input the data and location
step 3:create a newNode
step 4:newNode -> data = value
step 5:if(head == NULL) then
step 6:newNode -> previous = newNode -> next = NULL
step 7:head = newNode
step 8:else
step 9:initialize *temp1=head,temp2
step 10:while(temp1 -> data != location)
step 11:if(temp1 -> next == NULL) then
step 12:print Given node is not found in the list
step 13:else
step 14:temp1= temp1 -> next
step 15: temp2 = temp1 -> next
step 16:temp1 -> next = newNode
step 17:newNode -> previous = temp1
step 19:newNode -> next = temp2
step 20:temp2 -> previous = newNode
step 21:print insertion success
step 22:stop

Delete at beginning

step 1:start
step 2:if (head==NULL) then print list is empty and deletion is not possible
step 3:else initialize *temp=head
step 4:if(temp -> previous == temp -> next) then
step 5:head=NULL
step 6:free(temp)

step 7:else
step 8:head = temp -> next
step 9:head -> previous = NULL
step 10:free(temp)
step 11:print deletion success
step 12:stop

Delete at end

Step 1:start
step 2:if (head==NULL) then print list is empty and deletion is not possible
step 3:else initialize *temp=head
step 4:if(temp -> previous == temp -> next) then
step 5:head=NULL
step 6:free(temp)
step 7:else
step 8:while(temp-> next != NULL)
step 9:temp = temp -> next
step 10:temp -> previous -> next = NULL
step 11:free(temp)
step 12:print deletion success
step 13:stop

Delete a specific node

Step 1:start
step 2:if (head==NULL) then print list is empty and deletion is not possible
step 3:else initialize *temp=head
step 4:while(temp -> data != delValue)
step 5:if(temp -> next == NULL)
step 6:print Given node is not found in the list
step 7:else initialize temp=temp->next

step 8: if(temp == head)
step 9: head = NULL
step 10: free(temp)
step 11: else
step 12: temp -> previous -> next = temp -> next
step 13: free(temp)
step 14: print deletion success
step 15: stop

PROGRAM

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

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

void traverse()
{
    if (start == NULL) {
        printf("\nList is empty\n");
        return;
    }
    struct node* temp;
    temp = start;
    while (temp != NULL) {
        printf("Data = %d\n", temp->info);
        temp = temp->next;
    }
}

void insertAtFront()
{
    int data;
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->info = data;
    temp->prev = NULL;

    temp->next = start;
    start = temp;
}

void insertAtEnd()
{
    int data;
    struct node *temp, *trav;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->info = data;
    temp->next = NULL;
    trav = start;
```

```

if (start == NULL) {

    start = temp;
}

else {
    while (trav->next != NULL)
        trav = trav->next;
    temp->prev = trav;
    trav->next = temp;
}
}

void insertAtPosition()
{
    int data, pos, i = 1;
    struct node *temp, *newnode;
    newnode = malloc(sizeof(struct node));
    newnode->next = NULL;
    newnode->prev = NULL;
    printf("\nEnter position : ");
    scanf("%d", &pos);
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    newnode->info = data;
    temp = start;

    if (start == NULL) {
        start = newnode;
        newnode->prev = NULL;
        newnode->next = NULL;
    }

    else if (pos == 1) {
        newnode->next = start;
        newnode->next->prev = newnode;
        newnode->prev = NULL;
        start = newnode;
    }

    else {
        while (i < pos - 1) {
            temp = temp->next;
            i++;
        }
        newnode->next = temp->next;
        newnode->prev = temp;
        temp->next = newnode;
        temp->next->prev = newnode;
    }
}

```

```

    }
}

void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        start = start->next;
        if (start != NULL)
            start->prev = NULL;
        free(temp);
    }
}

void deleteEnd()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    temp = start;
    while (temp->next != NULL)
        temp = temp->next;
    if (start->next == NULL)
        start = NULL;
    else {
        temp->prev->next = NULL;
        free(temp);
    }
}

void deletePosition()
{
    int pos, i = 1;
    struct node *temp, *position;
    temp = start;

    if (start == NULL)
        printf("\nList is empty\n");

    else {
        printf("\nEnter position : ");
        scanf("%d", &pos);

        if (pos == 1) {
            position = start;

```

```

        start = start->next;
        if (start != NULL) {
            start->prev = NULL;
        }
        free(position);
        return;
    }

    while (i < pos - 1) {
        temp = temp->next;
        i++;
    }

    position = temp->next;
    if (position->next != NULL)
        position->next->prev = temp;
    temp->next = position->next;

    free(position);
}

int main()
{
    int choice;
    while (1) {

        printf("\n\t1 To see list\n");
        printf("\t2 For insertion at "
            " starting\n");
        printf("\t3 For insertion at "
            " end\n");
        printf("\t4 For insertion at "
            "any position\n");
        printf("\t5 For deletion of "
            "first element\n");
        printf("\t6 For deletion of "
            "last element\n");
        printf("\t7 For deletion of "
            "element at any position\n");
        printf("\t8 To exit\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                traverse();
                break;

```

```
    case 2:
        insertAtFront();
        break;
    case 3:
        insertAtEnd();
        break;
    case 4:
        insertAtPosition();
        break;
    case 5:
        deleteFirst();
        break;
    case 6:
        deleteEnd();
        break;
    case 7:
        deletePosition();
        break;

    case 8:
        exit(1);
        break;
    default:
        printf("Incorrect Choice. Try Again \n");
        continue;
    }
}
return 0;
}
```


OUTPUT

```
Activities Terminal Tue 12:59 #
anju@anju-pc: ~/Desktop
(base) anju@anju-pc:~/Desktop$ ./bit

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit

Enter Choice : 2
Enter number to be Inserted: 10

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit

Enter Choice : 3
Enter number to be Inserted: 50

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit

Enter Choice : 4
Enter position : 2
Enter number to be Inserted: 20

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
```

```
Activities Terminal Tue 12:59 #
anju@anju-pc: ~/Desktop
Enter Choice : 4
Enter position : 3
Enter number to be Inserted: 30

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit

Enter Choice : 4
Enter position : 4
Enter number to be Inserted: 40

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit

Enter Choice : 1
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50

1 To see list
2 For Insertion at starting
3 For Insertion at end
4 For Insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit

Enter Choice : 5
```

```
Activities Terminal Tue 12:59 anju@anju-pc ~/Desktop
File Edit View Search Terminal Help
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
1
Data = 20
Data = 30
Data = 40
Data = 50
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
6
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
1
Data = 20
Data = 30
Data = 40
Data = 948451968
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
7
Enter position : 3
1 To see list
```

```
Activities Terminal Tue 12:59 anju@anju-pc ~/Desktop
File Edit View Search Terminal Help
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
1
Data = 20
Data = 30
Data = 40
Data = 948451968
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
7
Enter position : 3
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
1
Data = 20
Data = 30
Data = 948451968
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter Choice :
```

AIM:Write a C program to construct a binary search tree and perform deletion, inorder traversal on it.

ALGORITHM

step 1:start
step 2:define a structure node
step 3:initialize value and pointers *l, *r,*t1,*t2 and assign *temp=NULL,*root=NULL
step 4:read the choice
step 5:if choice=1 then call insert()
step 6:if choice=2 then call delete()
step 7:if choice=3 then call inorder()
step 8:if choice=4 then call preorder()
step 9:if choice=5 then call postorder()
step 10:if choice=6 then exit
step 11:default print "wrong choice"
step 12:stop

To create a node

Step 1:start
Step 2:read value of node and assign to data
Step 3:temp->value=data
Step 4:temp->l=temp->r=NULL
Step 5:stop

insert a node in the tree

step 1:start
step 2:call create()
step 3:if(root== NULL) then root=temp
step 4:else
step 5:if ((temp->value > t->value) && (t->r != NULL)) then insert newnode right of
the root, search(t->r)

step 6:else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp

step 7:else if ((temp->value < t->value) && (t->l != NULL)) then insert newnode left
of the root, search(t->l)

step 8:else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp

step 9:stop

inorder traversal of tree

step 1: start

step 2:if(root==NULL) then exit no elements in the tree

step 3:if(t->l != NULL) then

step 4:inorder(t->l)

step 5:print t->value

step 6:if (t->r != NULL)

step 7:inorder(t->r)

step 8:stop

delete a node

step 1:start

step 2:if (root == NULL) then exit tree is empty

step 3:read the data to be deleted and assign to data

step 4:t1 = root,t2 = root

step 5:search1(root, data)

step 6:stop

preorder traversal of tree

step 1:start

step 2:if (root == NULL) then exit tree is empty

step 3:print t->value

step 4:if (t->l != NULL) then preorder(t->l)

step 5:if (t->r != NULL) then preorder(t->r)

step 6:stop

postorder traversal of tree

step 1:start

step 2:if (root == NULL) then exit tree is empty

step 3:if (t->l != NULL) then postorder(t->l)

step 4:if (t->r != NULL) then postorder(t->r)

step 5:print t->value

step 6:stop

PROGRAM

```
#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;
        }
    }
}

void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

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;
}

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
}

```



```

        else if ((temp->value < t->value) && (t->l != NULL))
            search(t->l);
        else if ((temp->value < t->value) && (t->l == NULL))
            t->l = temp;
    }
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);
}
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);
    }

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);
}

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);
}

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);
}
}

void delete1(struct btnode *t)
{
    int k;
    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
    }
}

```

```

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

```

```

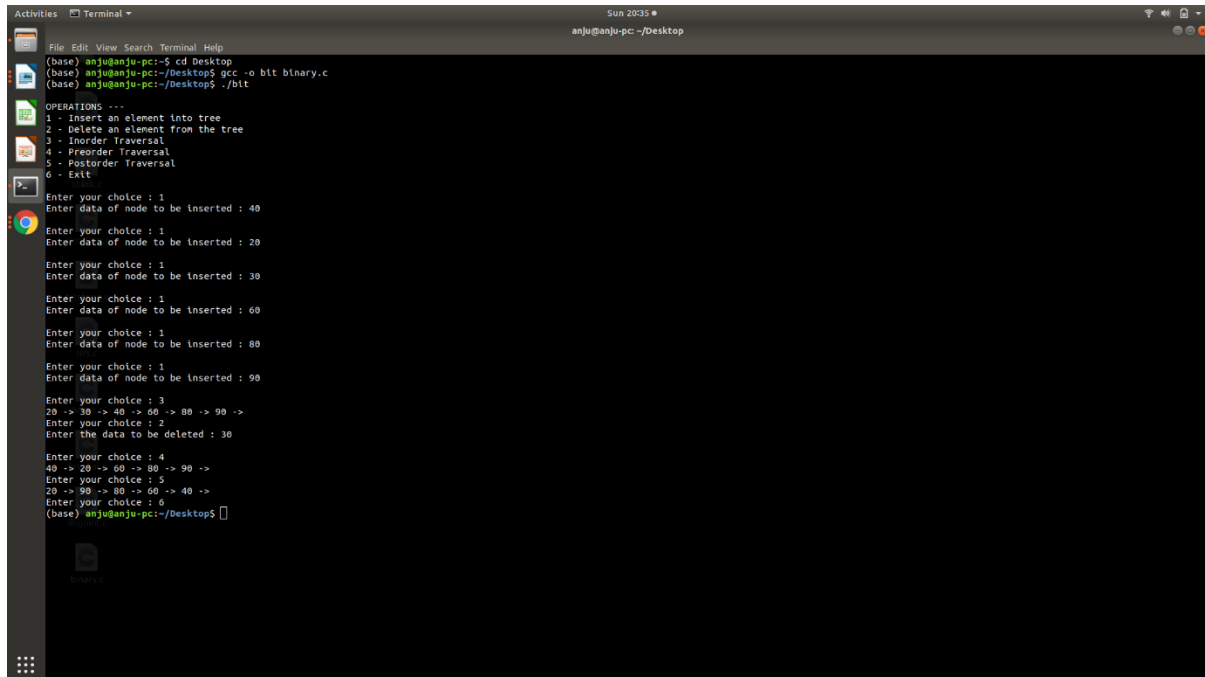
        t1->r = t->r;
    else
        t1->l = t->r;
    t == NULL;
    free(t);
    return;
}
else if ((t->l != NULL) && (t->r != NULL))
{
    t2 = root;
    if (t->r != NULL)
    {
        k = smallest(t->r);
        flag = 1;
    }
    else
    {
        k = largest(t->l);
        flag = 2;
    }
    search1(root, k);
    t->value = k;
}
}
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;

```

```
        return(smallest(t->l));
    }
    else
        return (t->value);
}

int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```

OUTPUT



```
Activities Terminal
Sun 20:35
anju@anju-pc: ~/Desktop
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit binary.c
(base) anju@anju-pc:~/Desktop$ ./bit

OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 - Exit

Enter your choice : 1
Enter data of node to be inserted : 40

Enter your choice : 1
Enter data of node to be inserted : 20

Enter your choice : 1
Enter data of node to be inserted : 30

Enter your choice : 1
Enter data of node to be inserted : 60

Enter your choice : 1
Enter data of node to be inserted : 80

Enter your choice : 1
Enter data of node to be inserted : 90

Enter your choice : 3
20 -> 30 -> 40 -> 60 -> 80 -> 90 ->
Enter your choice : 2
Enter the data to be deleted : 30

Enter your choice : 4
40 -> 20 -> 60 -> 80 -> 90 ->
Enter your choice : 5
20 -> 90 -> 80 -> 60 -> 40 ->
Enter your choice : 6
(base) anju@anju-pc:~/Desktop$
```

AIM:Write a C program to perform set operation using disjoint set

ALGORITHM

Step 1:start

Step 2:create a structure DisjSet with variables n,parent[10],rank[10]

Step 3:read number of elements and assign to n

Step 4:call makeset()

Step 5:print 1.union,2.find,3.display

Step 6:read a choice and assign to ch

Step 7:if ch==1 then read the element to perform union and call union(x,y)

Step 8:if ch==2 then read the element to check if connected components

Step 9:if(find(x)==find(y)) then print connected components

Step 10:else print not connected

Step 11:ch==3 then call display()

Step 12:stop

Makeset

Step 1:start

Step 2:for i=0 to n

Step 3:parent[i]=i;

Step 4:rank[i]=0

Step 5:stop

Display disjoint set

Step 1:start

Step 2:for i=0 to dis.n

Step 3:print parent

Step 4:for i=0 to dis.n

Step 5:print rank

Step 6:find() of given item x then

Step 7:if(parent[x]!=x) then

Step 8:parent[x]=find(parent[x])

Step 9:return parent[x]

Step 10:stop

Union of disjoint set

Step 1:start

Step 2:xset=find(x),yset=find(y)

Step 3:if(xset==yset) then return they already in same set

Step 4:else

Step 5:parent[yset]=xset then

Step 6: rank[xset] = rank[xset] + 1

Step 7:dis.rank[yset]=-1

Step 8:stop

PROGRAM

```
#include <stdio.h>

struct DisjSet {
    int parent[10];
    int rank[10];
    int n;
}dis;

void makeSet()
{
    for (int i = 0; i < dis.n; i++) {
        dis.parent[i] = i;
        dis.rank[i]=0;
    }
}

void displaySet()
{   printf("\nParent Array\n");
    for (int i = 0; i < dis.n; i++) {
        printf("%d ",dis.parent[i]); }
    printf("\nRank Array\n");
    for (int i = 0; i < dis.n; i++)
    {
        printf("%d ",dis.rank[i]);
    }
    printf("\n");
}

int find(int x)
{
    if (dis.parent[x] != x)
    {
        dis.parent[x] = find(dis.parent[x]);
    }
}
```

```

    }
    return dis.parent[x];
}
void Union(int x, int y)
{
    int xset = find(x);
    int yset = find(y);
    if (xset == yset)
        return;
    if (dis.rank[xset] < dis.rank[yset])
    {
        dis.parent[xset] = yset;
        dis.rank[xset] = -1;
    }
    else if (dis.rank[xset] > dis.rank[yset])
    {
        dis.parent[yset] = xset;
        dis.rank[yset] = -1;
    }
    else {
        dis.parent[yset] = xset;
        dis.rank[xset] = dis.rank[xset] + 1;
        dis.rank[yset] = -1;
    }
}

int main()
{
    int n,x,y;
    printf("How many elements ?");
    scanf("%d",&dis.n);

```

```

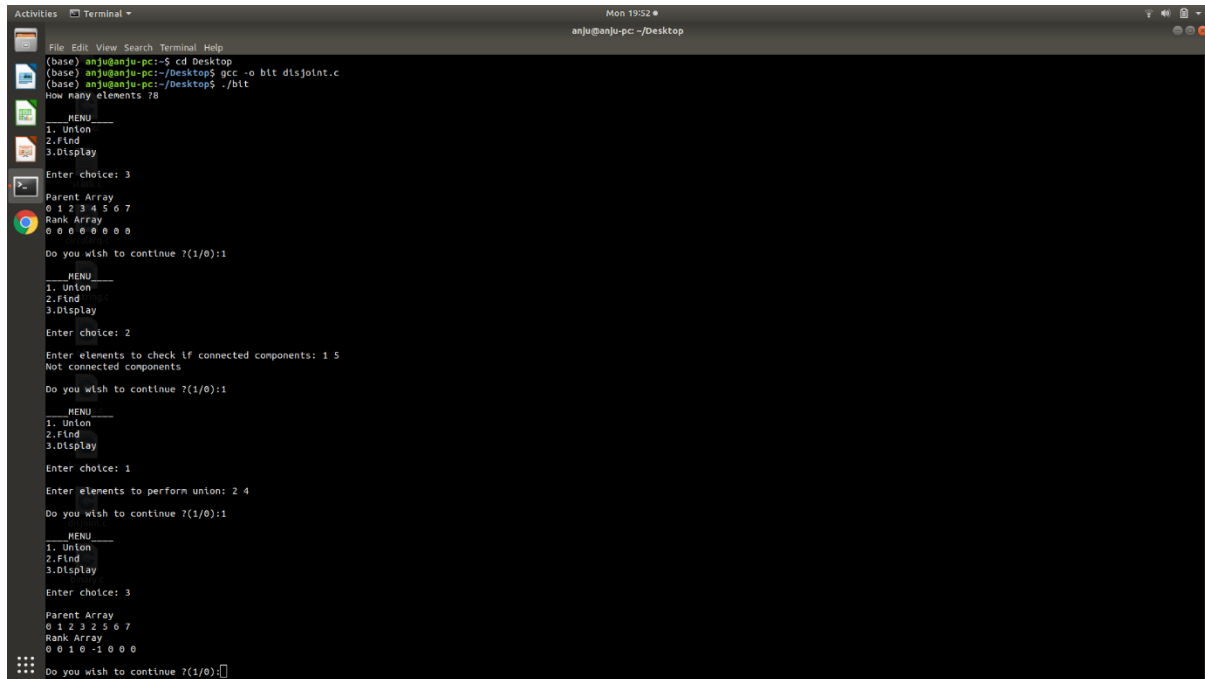
makeSet();

int ch,wish;

do
{
printf("\n____MENU____\n");
printf("1. Union \n2.Find\n3.Display\n");
printf("enter choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter elements to perform union");
scanf("%d %d",&x,&y);
Union(x, y);
break;
case 2: printf("Enter elements to check if connected components");
scanf("%d %d",&x,&y);
if (find(x) == find(y))
printf("Connected components\n") ;
else
printf("Not onnected components \n") ;
break;
case 3: displaySet();
break;
}
printf("\nDo you wish to continue?(1/0)\n");
scanf("%d",&wish);
}while(wish==1);
return 0;
}

```

OUTPUT



```
Mon 19:52
anju@anju-pc: ~/Desktop
File Edit View Search Terminal Help
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit disjoint.c
(base) anju@anju-pc:~/Desktop$ ./bit
How many elements 78
MENU
1. Union
2. Find
3. Display
Enter choice: 3
Parent Array
0 1 2 3 4 5 6 7
Rank Array
0 0 0 0 0 0 0 0
Do you wish to continue ?(1/0):1
MENU
1. Union
2. Find
3. Display
Enter choice: 2
Enter elements to check if connected components: 1 5
Not connected components
Do you wish to continue ?(1/0):1
MENU
1. Union
2. Find
3. Display
Enter choice: 1
Enter elements to perform union: 2 4
Do you wish to continue ?(1/0):1
MENU
1. Union
2. Find
3. Display
Enter choice: 3
Parent Array
0 1 2 3 2 5 6 7
Rank Array
0 0 1 0 -1 0 0 0
Do you wish to continue ?(1/0):
```

PROGRAM NUMBER :07

Date:13/12/2021

AIM: Write a C program to perform set operation using bit strings

ALGORITHM

Step 1:start

Step 2:print 1. Input,2.union,3.intersection,4.difference

Step 3:read a choice and assign to ch

Step 4:if ch==1 then call input()

Step 5:if ch==2 then call union()

Step 6:if ch==3 then call intersection()

Step 7:if ch==4 then call difference()

Step 8:else exit

Step 9:stop

Input element to bit string

Step 1:start

Step 2:read the size of first set and assign to n

Step 3:for i=0 to n read the elements

Step 2:read the size of second set and assign to n

Step 3: for i=0 to n read the elements

Step 4:print the first and second

Step 5:stop

Display the elements in bit string

Step 1:start

Step 2:print third set for i=0 to 9

Step 3:if c[i]=0 then print i+1

Step 4: stop

Union in bit string

Step 1:start

Step 2:for i=0 to 9

Step 3:if $a[i] \neq b[i]$ then $c[i]=1$

Step 4:else $c[i]=a[i]$

Step 5:call display()

Step 6:stop

Intersection in bit string

Step 1:start

Step 2:for i=0 to 9

Step 3:if $a[i]=b[i]$ then $c[i]=1$

Step 4:else $c[i]=0$

Step 5:display third set

Step 6:for i=0 to 9 print $c[i]$

Step 7:call display()

Step 8:stop

Difference in bit string

Step 1:start

Step 2:for i=0 to 9

Step 3:if $a[i]==1 \ \&\& \ b[i]==1$ then $c[i]=1$

Step 4:else $c[i]=0$

Step 5: for i=0 to 9 print $c[i]$

Step 6:call display()

Step 7:stop

PROGRAM

```
//Universal Set is { 1,2,3,4,5,6,7,8,9}

#include <stdio.h>

void input();
void output();
void setunion();
void intersection();
void difference();

int a[]={0,0,0,0,0,0,0,0,0},b[]={0,0,0,0,0,0,0,0,0} ;

int main()
{
    int ch,wish;
    do
    {
        printf("\n____MENU____\n");
        printf("1.Input\n2.Union\n3.Intersection\n4.Difference\n");
        printf("enter choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:input();
                    break;
            case 2:setunion();
                    break;
            case 3:intersection();
                    break;
            case 4:difference();
                    break;
        }
    }
```



```
printf("\nDo you wish to continue ?(1/0)\n");  
scanf("%d",&wish);
```

```
}while(wish==1);
```

```
}
```

```
void input()
```

```
{ int n,x,i;
```

```
printf("enter size of the 1st set\n");
```

```
scanf("%d",&n);
```

```
printf("enter elements\n");
```

```
for(i=0;i<n;i++)
```

```
{ scanf("%d",&x);
```

```
    a[x-1]=1;
```

```
}
```

```
printf("enter size of the 2nd set\n");
```

```
scanf("%d",&n);
```

```
printf("enter elements\n");
```

```
for(i=0;i<n;i++)
```

```
{ scanf("%d",&x);
```

```
    b[x-1]=1;
```

```
}
```

```
printf("\n1st set\n");
```

```
for(i=0;i<9;i++)
```

```
{ printf("%d",a[i]);
```

```
}
```

```
printf("\n2nd set\n");
```

```
for(i=0;i<9;i++)
```

```
{ printf("%d",b[i]);
```

```

    }
}

void output(int c[])
{
    int i;
    printf("\n Set is");
    for(i=0;i<9;i++)
    {
        if (c[i]!=0)
            printf(" %d ",i+1);

    }
}

void setunion()
{
    int i,c[10];
    for(i=0;i<9;i++)
    {
        if (a[i]!=b[i])
            c[i]=1;
        else c[i]=a[i];
    }
    for(i=0;i<9;i++)
    {
        printf("%d",c[i]);
    }
    output(c);
}

void intersection()
{
    int i,c[10];
    for(i=0;i<9;i++)
    {
        if (a[i]==b[i])
            c[i]=a[i];
        else c[i]=0;
    }
}

```

```

    }
    for(i=0;i<9;i++)
    { printf("%d",c[i]);
    }
    output(c);
}

void difference()
{int i,c[10];
  for(i=0;i<9;i++)
  { if (a[i]==1 && b[i]==0)
    c[i]=1;
    else c[i]=0;
  }
  for(i=0;i<9;i++)
  { printf("%d",c[i]);
  }
  output(c);

}

```

OUTPUT

```
Activities Terminal Sat 22:49
anju@anju-pc ~/Desktop

1.Input
2.Union
3.Intersection
4.Difference
enter choice: 1
enter size of the 1st set: 5
enter elements:
1
2
3
4
5
enter size of the 2nd set: 5
enter elements:
4
5
6
7
8
1st set
111110000
2nd set
000111110
Do you wish to continue ?(1/0): 1

MENU
1.Input
2.Union
3.Intersection
4.Difference
enter choice: 2
111111110
Set is 1 2 3 4 5 6 7 8
Do you wish to continue ?(1/0): 1

MENU
1.Input
2.Union
3.Intersection
4.Difference
enter choice: 3
000110000
Set is 4 5
Do you wish to continue ?(1/0): 1

MENU
1.Input
2.Union
3.Intersection
4.Difference
enter choice: 4
111000000
Set is 1 2 3
Do you wish to continue ?(1/0):
```

PROGRAM NUMBER :08

Date:03/01/2022

AIM:Write a C program to implement BFS

ALGORITHM

Step 1:start

Step 2:enter the number of vertices

Step 3:read the graph in matrix form

Step 4:enter the starting vertex

Step 5:repeat the step4&5 until the queue is empty

Step 6:take the front item of the queue and add it to the visited list

Step 7:create a list of that vertex adjacent node.Add the one which aren't in the

Visited list to the back of the queue

Step 8:stop

PROGRAM

```
#include<stdio.h>

#include<conio.h>

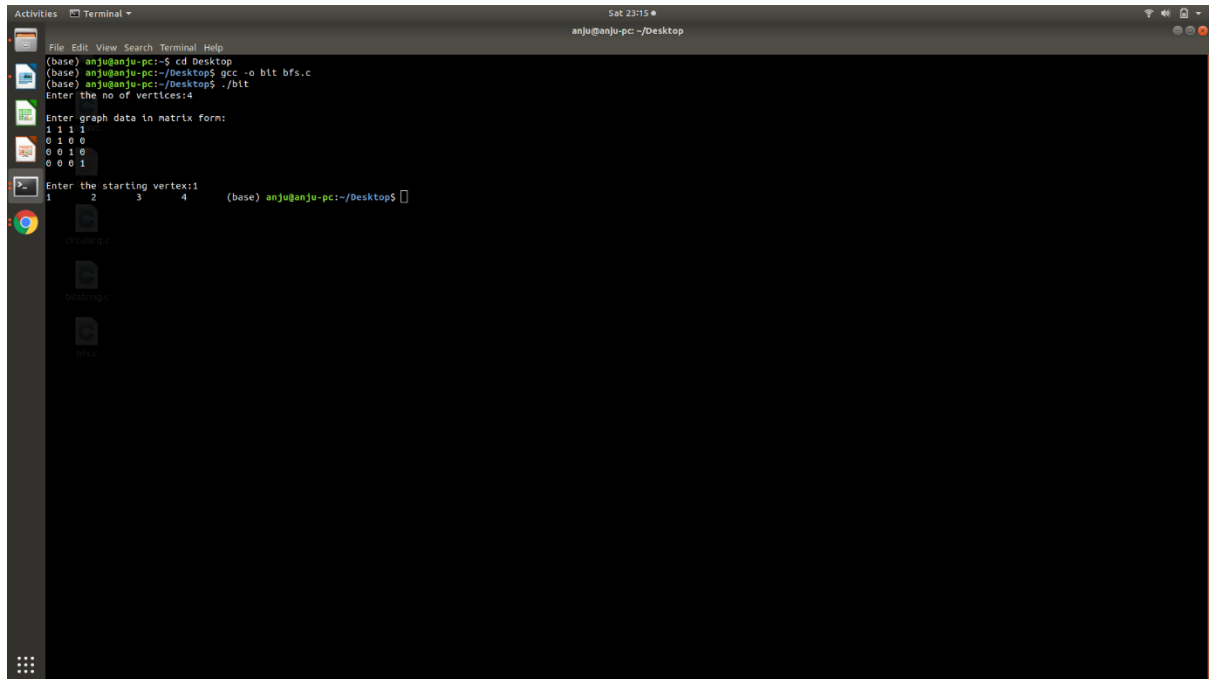
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)
{
    for(i=1;i<=n;i++)
        if(a[v][i] && !visited[i])
            q[++r]=i;
    if(f<=r)
    {
        visited[q[f]]=1;
        printf("%d\t",q[f]);
        bfs(q[f++]);
    }
}

void main()
{
    int v;
    clrscr();
    printf("Enter the no of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        q[i]=0;
        visited[i]=0;
    }
    printf("\nEnter graph data in matrix form:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
```

```
scanf("%d",&a[i][j]);  
printf("\nEnter the starting vertex:");  
scanf("%d",&v);  
visited[v]=1;  
printf("%d",v);  
bfs(v);  
getch();  
}
```

OUTPUT



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sat 23:15, anju@anju-pc: ~/Desktop). The terminal shows the following commands and output:

```
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit bfs.c
(base) anju@anju-pc:~/Desktop$ ./bit
Enter the no of vertices:4
Enter graph data in matrix form:
1 1 1 1
0 1 0 0
0 0 1 0
0 0 0 1
Enter the starting vertex:1
1 2 3 4 (base) anju@anju-pc:~/Desktop$
```

The terminal window has a sidebar on the left with icons for Activities, Terminal, and a file manager. The file manager shows a directory structure with folders named "Desktop", "Downloads", "Music", and "Videos".

AIM: Write a C program to implement DFS

ALGORITHM

Step 1:start

Step 2:create a structure node with element vertex and *next

Step 3:read number of vertices and assign to v

Step 4:initialize adj[] with a null

Step 5:read edges and insert them in adj[]

Step 6:represent the edges into adjacency list

Step 7:acquire only for the new node

Step 8:insert the node in the new node

Step 9:go to end of the linked list

Step 10:keep an array visited[] to keep track of the visited vertices

Step 11:if visited[i]=1 represent that vertex is has been visited before and the DFS function
for some already visited node need not be called

Step 12:repeat the step until all the vertex are visited

Step 13:print the vertex

Step 14:stop

PROGRAM

```
#include<stdio.h>

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

int v,e;
struct node **adj,**adj1;
int *que,*visited,*ft;
int f=-1,r=-1,t=0;
void dfs();
void dfsvisit(int);
void dfs1();
void dfsvisit1(int);
void adjlisRep(struct node **adj,int s,int en)
{
    struct node *ne=(struct node*)malloc(sizeof(struct node));
    ne->vertex=en;
    ne->next=adj[s];
    adj[s]=ne;
}

void main()
{
    int s,i,en;
    struct node *ptr;
    printf("Enter number of vertices:");
    scanf("%d",&v);
    adj=(struct node **)malloc((v+1)*sizeof(struct node *));
    adj1=(struct node **)malloc((v+1)*sizeof(struct node *));
    for(i=0;i<=v;i++)
```

```

adj[i]=adj1[i]=NULL;
printf("Enter number of edges:");
scanf("%d",&e);
printf("Enter the edges:");
printf("Start End\n");
for(i=0;i<e;i++)
{
    scanf("%d%d",&s,&en);
    adjlisRep(adj,s,en);
    adjlisRep(adj1,en,s);
}
dfs();
dfs1();
}
void dfs()
{
    int i;
    visited=(int *)malloc((v+1)*sizeof(int));
    ft=(int *)malloc((v+1)*sizeof(int));
    for(i=0;i<=v;i++)
        visited[i]=0;
    printf("dfs");
    for(i=1;i<=v;i++)
        if(visited[i]==0)
            dfsvisit(i);
}
void dfsvisit(int u)
{
    int w;
    struct node *ptr;

```

```

        visited[u]=1;
        printf("%d",u);
        ptr=adj[u];
        +while(ptr!=NULL)
        {
            w=ptr->vertex;
            if(visited[w]==0)
                dfsvisit(w);
            ptr=ptr->next;
        }
        t++;
        ft[u]=t;
    }
void dfsvisit1(int u)
{
    int w;
    struct node *ptr;
    visited[u]=1;
    printf("%d",u);
    ptr=adj1[u];
    while(ptr!=NULL)
    {
        w=ptr->vertex;
        if(visited[w]==0)
            dfsvisit1(w);
        ptr=ptr->next;
    }
}
int visitedAll()
{

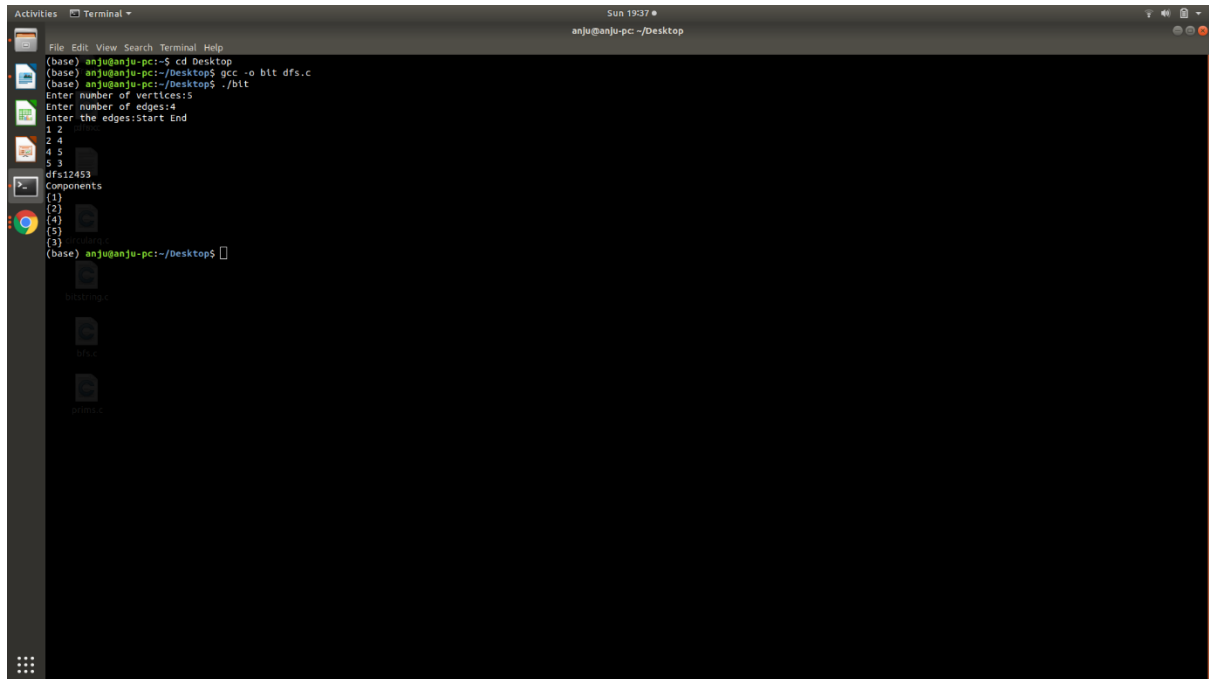
```

```

        int i,flag=1;
        for(i=1;i<=v;i++)
            if(visited[i]==0)
            {
                flag=0;
                break;
            }
        return flag;
    }
void dfs1()
{
    int i,max=0,ver;
    printf("\nComponents\n");
    for(i=0;i<=v;i++)
        visited[i]=0;
    while(!visitedAll())
    {
        max=0;
        for(i=1;i<=v;i++)
        {
            if(visited[i]==0 && ft[i]>max)
            {
                ver=i;
                max=ft[i];
            }
        }
        printf("{");
        dfsvisit1(ver);
        printf("}\n");
    }
}

```

OUTPUT



A terminal window titled "Sun 19:37" and "anju@anju-pc: ~/Desktop" showing the execution of a Depth-First Search (DFS) algorithm. The user enters the following commands and receives the following output:

```
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit dfs.c
(base) anju@anju-pc:~/Desktop$ ./bit
Enter number of vertices:5
Enter number of edges:4
Enter the edges:Start End
1 2
2 4
4 5
5 3
dfs:2453
Components
(1)
(2)
(4)
(5)
(3)
(base) anju@anju-pc:~/Desktop$
```

The output shows the DFS traversal order as 2453 and the components of the graph as (1), (2), (4), (5), and (3).

PROGRAM NUMBER :10

Date:17/01/2022

AIM: Write a C program to create MST using Prim's algorithm

ALGORITHM

Step 1:start

Step 2:read the number of nodes and assign to n

Step 3:read the adjacency matrix

Step 4: if(cost[i][j]==0) then cost[i][j]=999

Step 5: visited[1]=1 and print node

Step 6:while(ne<n) then

Step 7:traverse adjacency matrix

Step 8:if cost[i][j]<min and if visited[i]!=0 then

Step 9:min=cost[i][j] ,a=u,I,b=v=j

Step 10:if visited[u]==0 or visited[v]==0

Step 11:print edge

Step 12:mincost+=min

Step 13:visited[b]=1

Step 14:cost[a][b]=cost[b][a]=999

Step 15:print Minimum cost

Step 16:stop

PROGRAM

```
#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,n=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

    clrscr();

    printf("\nEnter the number of nodes:");

    scanf("%d",&n);

    printf("\nEnter the adjacency matrix:\n");

    for(i=1;i<=n;i++)

    for(j=1;j<=n;j++)

    {

        scanf("%d",&cost[i][j]);

        if(cost[i][j]==0)

            cost[i][j]=999;

    }

    visited[1]=1;

    printf("\n");

    while(ne<n)

    {

        for(i=1,min=999;i<=n;i++)

        for(j=1;j<=n;j++)

        if(cost[i][j]<min)

        if(visited[i]!=0)

        {

            min=cost[i][j];

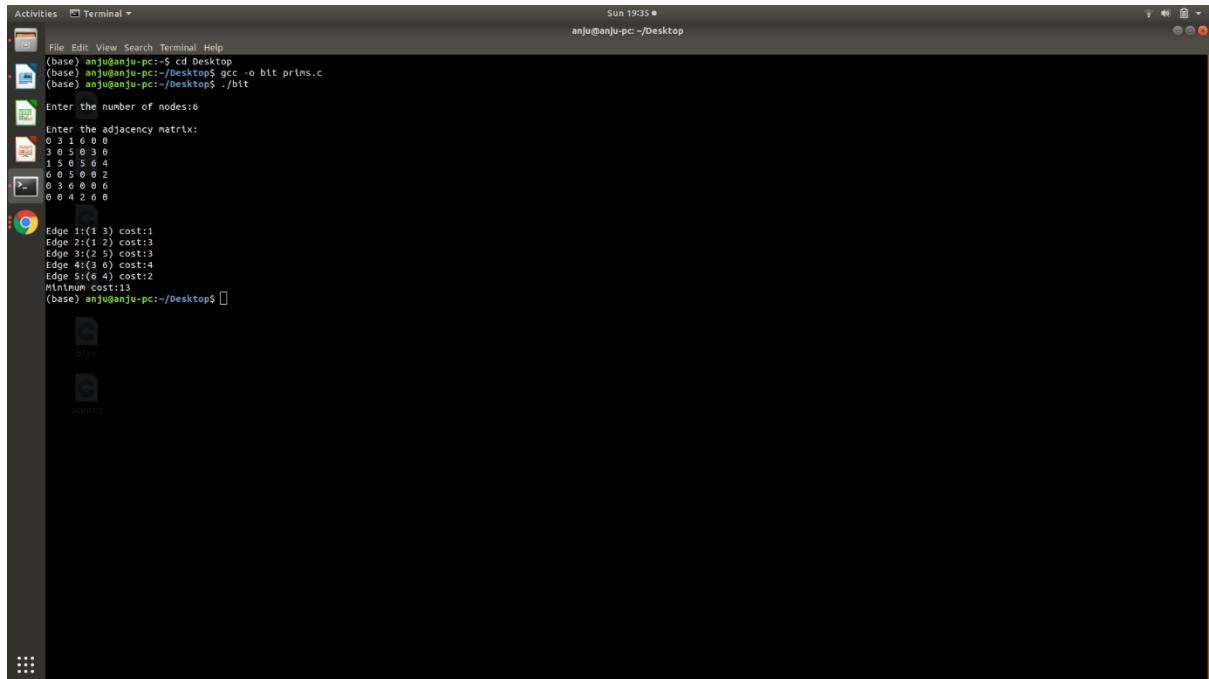
            a=u=i;

            b=v=j;
```



```
    }
    if(visited[u]==0||visited[v]==0)
    {
        printf("\nEdge %d:(%d %d) cost:%d",ne++,a,b,min);
        mincost+=min;
        visited[b]=1;
    }
    cost[a][b]=cost[b][a]=999;
}
printf("\nMinimum cost:%d\n",mincost);
}
```

OUTPUT



A terminal window titled "Sun 19:35" and "anju@anju-pc: ~/Desktop" displays the following output:

```
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit prims.c
(base) anju@anju-pc:~/Desktop$ ./bit

Enter the number of nodes:6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
0 6 5 6 0 2
0 3 6 6 0 6
0 0 4 2 6 0

Edge 1:(1 3) cost:1
Edge 2:(1 2) cost:3
Edge 3:(2 5) cost:3
Edge 4:(3 0) cost:4
Edge 5:(6 4) cost:2
Minimum cost:13
(base) anju@anju-pc:~/Desktop$
```

PROGRAM NUMBER :11

Date:26/01/2022

AIM: Write a C program to implement topological sorting

ALGORITHM

Step 1:start

Step 2:read number of vertices in the graph and assign to n

Step 3:read adjacency matrix

Step 4:find indegree of each vertex

Step 5:set count=0

Step 6:add vertex v to array and increment count

Step 7:delete all edges going from vertex v

Step 8:if count<n then print no topological ordering possible,graph contain cycle

Step 9:else print array ,topological order of graph

Step 10:stop

PROGRAM

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int n;

int adj[MAX][MAX];

void create_graph();

int queue[MAX], front = -1, rear = -1;

void insert_queue(int v);

int delete_queue();

int isEmpty_queue();

int indegree(int v);

int main()
{
    int i,v,count,topo_order[MAX],indeg[MAX];

    create_graph();

    for(i=1;i<=n;i++)
    {
        indeg[i] = indegree(i);
        if( indeg[i] == 0 )
            insert_queue(i);
    }

    count = 0;

    while( !isEmpty_queue( ) && count < n )
    {
        v = delete_queue();
        topo_order[++count] = v;
        for(i=1; i<=n; i++)
        {
            if(adj[v][i] == 1)
```

```

        {
            adj[v][i] = 0;
            indeg[i] = indeg[i]-1;
            if(indeg[i] == 0)
                insert_queue(i);
        }
    }
}

if( count < n )
{
    printf("\nNo topological ordering possible, graph contains cycle\n");
    exit(1);
}

printf("\nVertices in topological order are :\n");
for(i=1; i<=count; i++)
    printf( "%d ",topo_order[i] );
printf("\n");
return 0;
}

void insert_queue(int vertex)
{
    if (rear == MAX-1)
        printf("\nQueue Overflow\n");
    else
    {
        if (front == -1) /*If queue is initially empty */
            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 del_item;
    if (front == -1 || front > rear)
    {
        printf("\nQueue Underflow\n");
        exit(1);
    }
    else
    {
        del_item = queue[front];
        front = front+1;
        return del_item;
    }
}
int indegree(int v)
{
    int i,in_deg = 0;
    for(i=1; i<=n; i++)
        if(adj[i][v] == 1)
            in_deg++;
    return in_deg;
}
void create_graph()

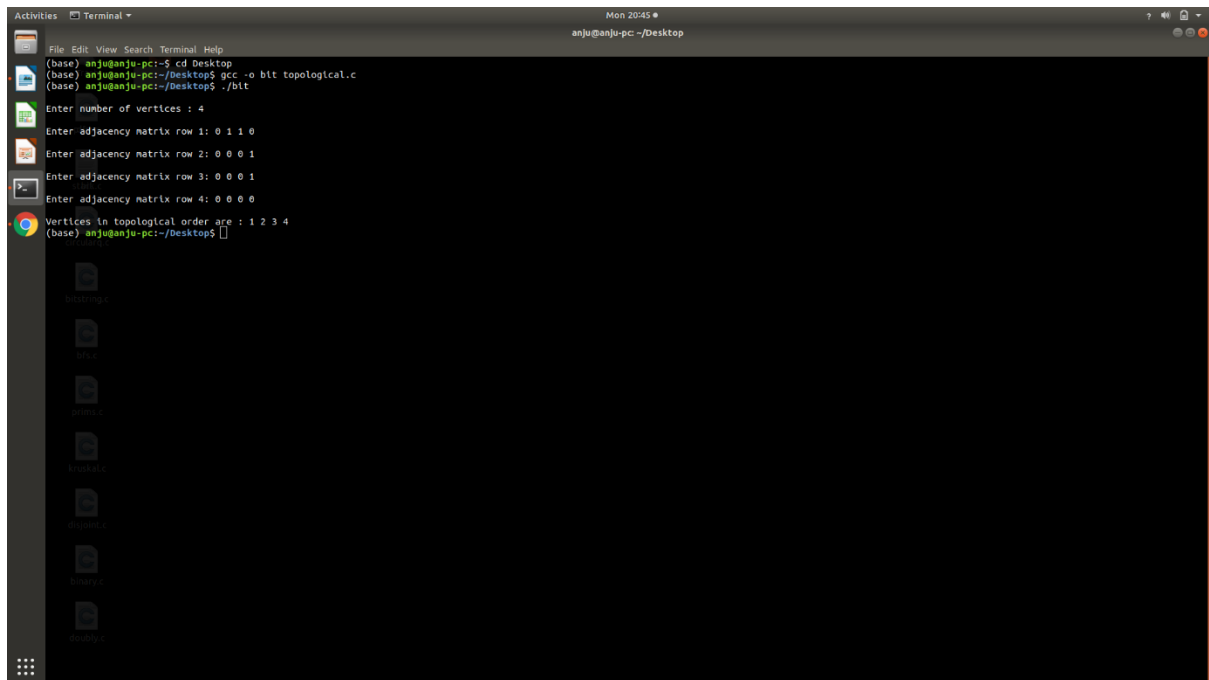
```

```

{
    int i,j,max_edges,origin,destin;
    printf("\nEnter number of vertices : ");
    scanf("%d",&n);
    max_edges = n*(n-1);
    for(i=1; i<=n; i++)
    { printf("\nEnter adjcency matrix row %d: \n",i);
        for(j=1;j<=n;j++)
        {
            scanf("%d",&adj[i][j]);
        }
    }
}

```

OUTPUT



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Mon 20:45, anju@anju-pc ~/Desktop). The terminal shows the following commands and output:

```
(base) anju@anju-pc:~$ cd Desktop
(base) anju@anju-pc:~/Desktop$ gcc -o bit topological.c
(base) anju@anju-pc:~/Desktop$ ./bit

Enter number of vertices : 4
Enter adjacency matrix row 1: 0 1 1 0
Enter adjacency matrix row 2: 0 0 0 1
Enter adjacency matrix row 3: 0 0 0 1
Enter adjacency matrix row 4: 0 0 0 0

Vertices in topological order are : 1 2 3 4
(base) anju@anju-pc:~/Desktop$
```

The terminal window also displays a sidebar with icons for various applications and files, including a file manager, a text editor, and several image files.