# Faculty Orientation Workshop
# on
# Data Structures

Under the Aegis of BoS (E&TC), SPPU, Pune
SE E&TC/ Electronics) 2019 Course
(22nd to 26th June 2020)

Mrs.Rajeshwari.M.Thadi(HOD E&TC SKNSITS,LONAVALA)

# Data Structures Lab

**Program-4** Create a singly linked list with options:

- Insert (at front, at end, in themiddle),
- Delete (at front, at end, in themiddle),
- Display,
- Revert the SLL

## Algorithm

**Start:**

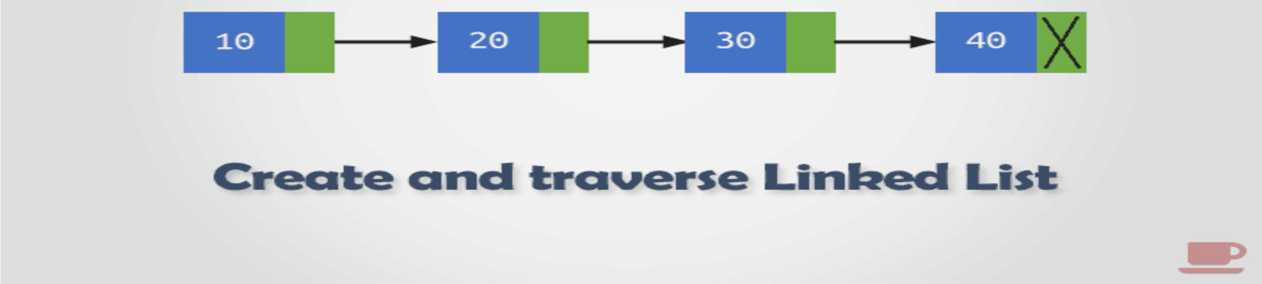
**Step1**. MENU DRIVEN PROGRAM FOR SINGLE LINKED LIST OPERATIONS
1.CREATE
2.DISPLAY
3.INSERT AT BEGINNING
4.INSERT AT SPECIFIED POSITION
5.DELETE FROM BEGINNING
6.DELETE FROM SPECIFIED POSITION
7.EXIT

**Step-2** Creation of structure

```
struct node {

 int data;

struct node *next;

 };
```

**Step-3** Declare a structure variable

struct node *start=NULL; & also decide the number of nodes in linked list



```
Create()
Step 1 Declaration of variables
        struct node *temp,*ptr;
Step-2  Allocat memory dynamically using malloc function
temp=(struct node *)malloc(sizeof(struct node));
Step-3  Check if node temp==NULL if yes print memory not allocated else take the data from keyboard using scanf() function.
Step 4 Check if start==Null if yes  start=temp;
                Else  ptr=start;
             while(ptr->next!=NULL)
             {
                  ptr=ptr->next;
             }
             ptr->next=temp;
Step-5 Repeat the step-4
```

```
Display ()
Step 1 Declaration of variables
          struct node *ptr;
Step 2 Check if node temp==NULL if yes print empty list
      else  ptr=start;
          printf("\nThe List elements are:\n");
          while(ptr!=NULL)
          {
               printf("%d\t",ptr->info );
               ptr=ptr->next ;
          }
```

Insert_at_start()

**Step 1** Declaration of variables

```
        struct node *temp,*ptr;
```

**Step-2** Allocat memory dynamically using malloc function

```
temp=(struct node *)malloc(sizeof(struct node));
```

**Step-3** check if node temp==NULL if yes print memory not allocated

Else Take the data from keyboard using scanf() function

**Step-4** if(start==NULL)

```
       {
              start=temp;
       }
       else
       {
              temp->next=start;
              start=temp;
       }
```

Insert_at_ anyposition()

**Step1** Declaration of variables

```
        struct node *temp,*ptr;
          int i,pos;
```

**Step-2** Allocat memory dynamically using malloc function

```
             temp=(struct node *)malloc(sizeof(struct node));
```

**Step-3** Check if node temp==NULL if yes print memory not allocated

Else take the data from keyboard using scanf() function

**Step-4** Also take the position from keyboard using scanf() function

```
                    if(pos==0)
                      {
                  temp->next=start;
                   start=temp;
                     }
```

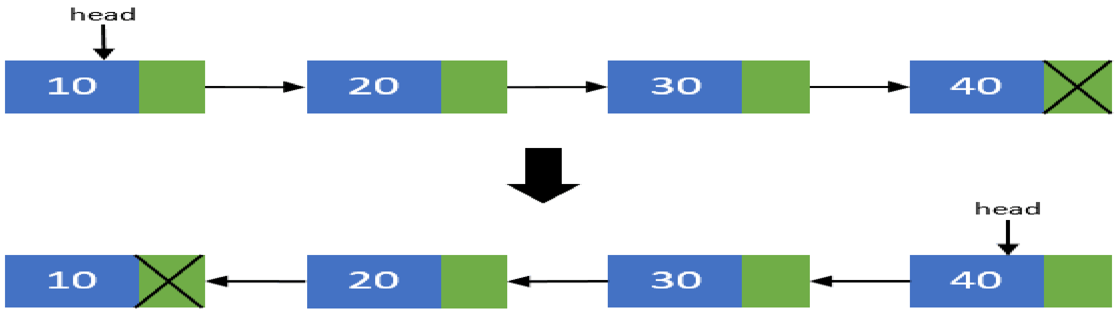
**Step-5**

```
                   for(i=0,ptr=start;i<pos-1;i++)
                      {
                 ptr=ptr->next;
                 if(ptr==NULL)
                 {
                        printf("\nPosition not found:[Handle with care]\n");
                        return;
                 }
          }
          temp->next =ptr->next ;
          ptr->next=temp;
```

```
Delete_at_begin()
 Step 1 Declaration of variables
             struct node *temp,*ptr;
Step-2  Allocat memory dynamically using malloc function
            temp=(struct node *)malloc(sizeof(struct node));
Step-3  Check if node temp==NULL if yes print list is empty
Step-4  ptr=start;
              start=start->next ;
              printf("\nThe deleted element is :%d\t",ptr->info);
              free(ptr);
```

```
Delete_at_anyposition()
Step1  Declaration of variables
          struct node *temp,*ptr;
           int i,pos;
Step-2  Allocat memory dynamically using malloc function
           temp=(struct node *)malloc(sizeof(struct node));
Step-3  Check if node temp==NULL if yes print memory not alloca
Else Choose the position of the node to be deleted.
Step-4              ptr=start;
                   for(i=0;i<pos;i++)
                   {
                          temp=ptr;
                          ptr=ptr->next ;
                          if(ptr==NULL)
                          {
                                 printf("\nPosition not Found:\n");
                                 return;
                          }
                   }
                   temp->next =ptr->next ;
                   printf("\nThe deleted element is:%d\t",ptr->info );
                    free(ptr);
```

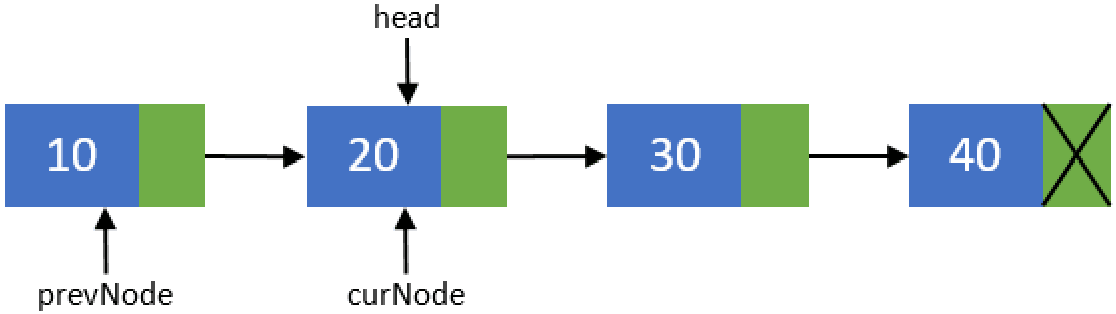
# Reverse SLL



**Step-1** prevNode = head

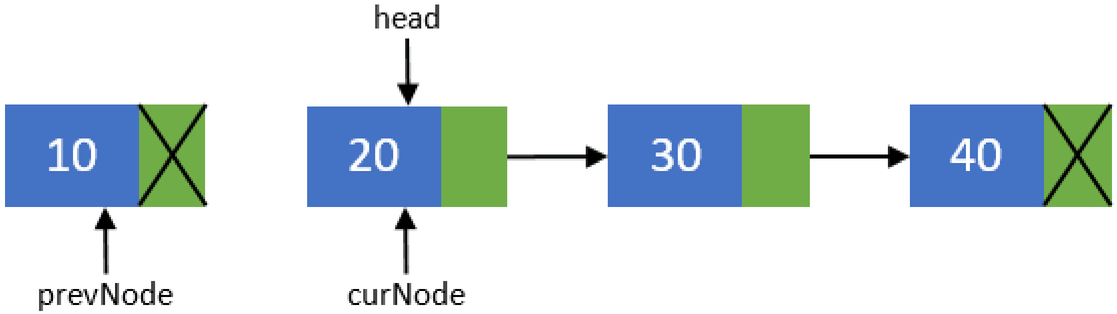
head = head->next

curNode = head.



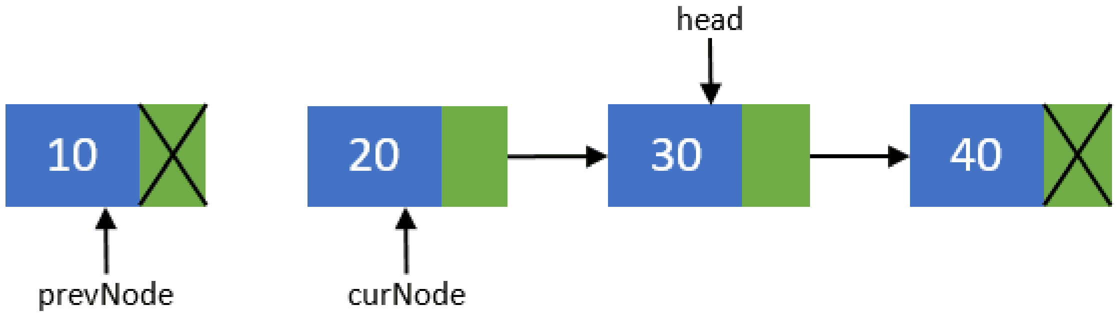
**Step-2**

**prevNode->next = NULL**



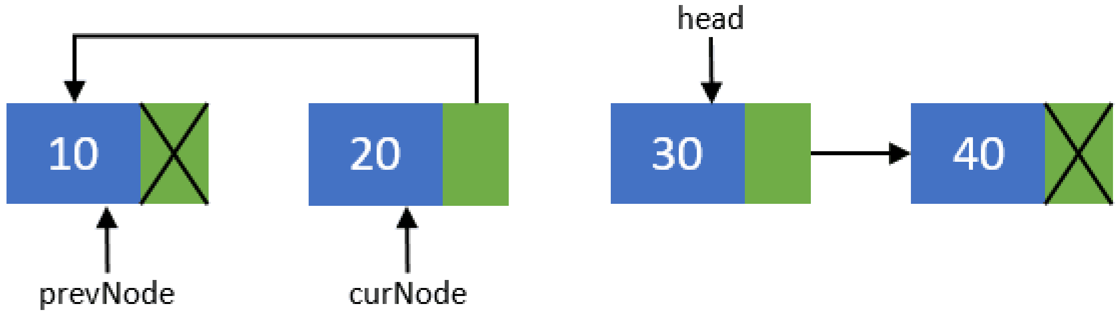
**Step-3**

head = head->next



**Step-4**

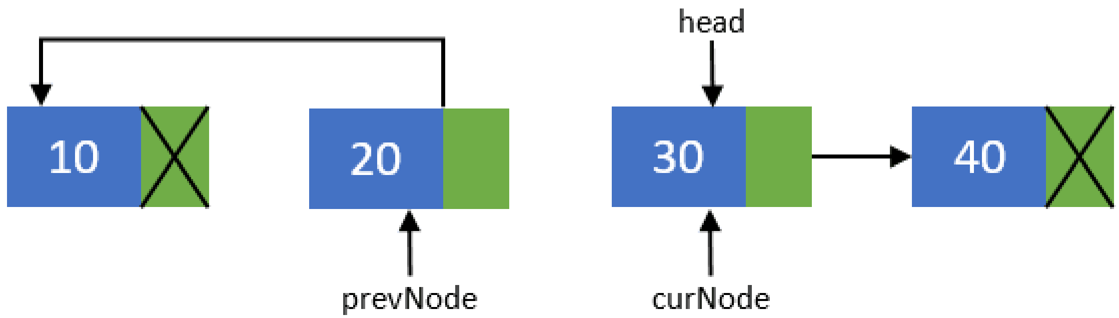
curNode->next = prevNode



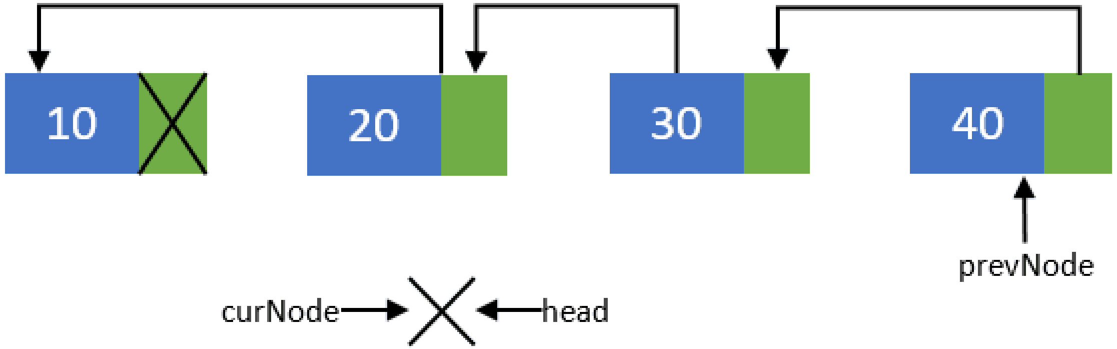
**Step-5**

prevNode = curNode;

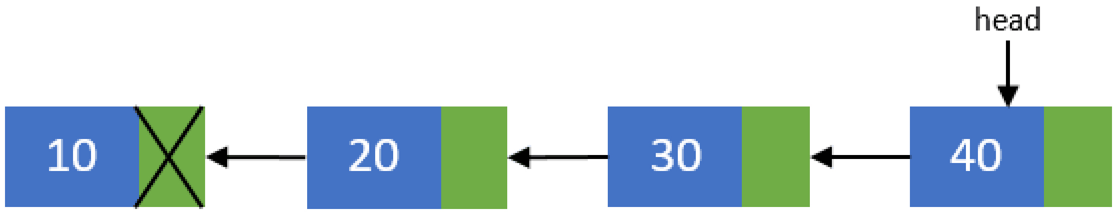
curNode = head.



**Step 6** Repeat steps 3-5 till head pointer becomes NULL

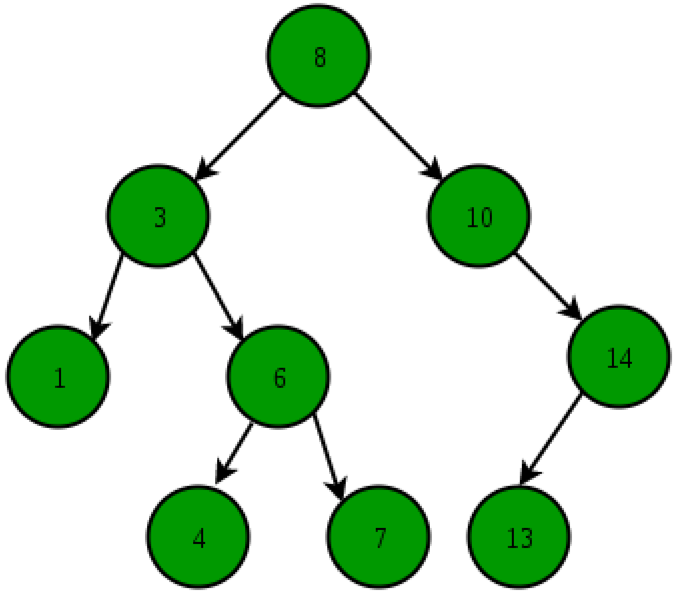


head = prevNode



## Program -5 Implement Binary search tree with operations Create, search, and recursive traversal

Treetraversals.mp4



**Start**

**Step 1** Create a structure

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

**Step 2**. MENU DRIVEN PROGRAM FOR SINGLE LINKED LIST OPERATIONS

1.INSERT
2.SEARCH
3.INORDER
4.PREORDER
5.POSTORDER
6.EXIT

**Step-3** Allocat memory dynamically using malloc function

```
nw=(struct node *)malloc(sizeof(struct node));
```

**Step-4** Check if node temp==NULL if yes print memory not allocated

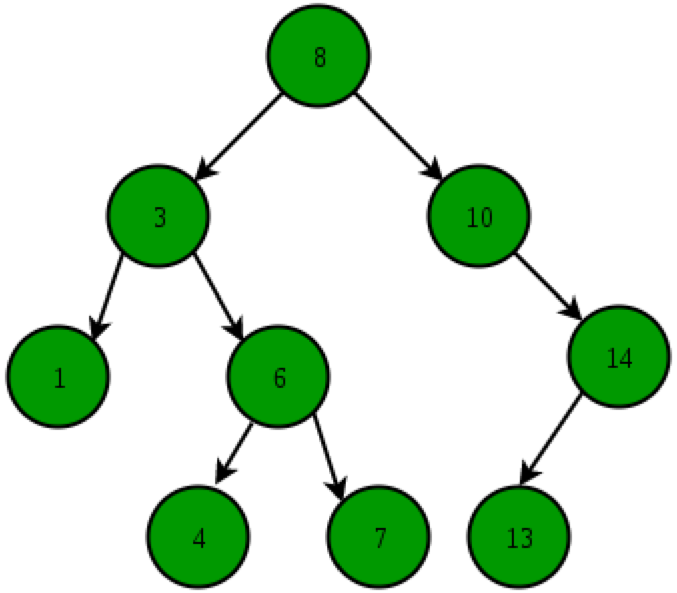
Else take the data from keyboard using scanf() function

**Step-5**

```
if(root==NULL)
        root=nw;
    else
        insert(root,nw);
```

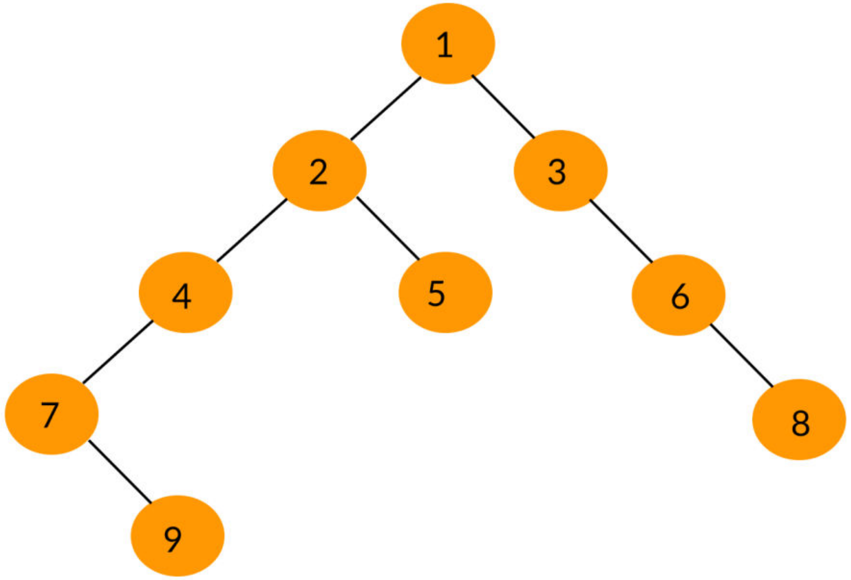
**llustration to search 6 in above tree:**

1. Start from root.
2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false



```
// C function to search a given key in a given BST
struct node* search(struct node* root, int key)
{
   // Base Cases: root is null or key is present at root
   if (root == NULL || root->key == key)
      return root;

   // Key is greater than root's key
   if (root->key < key)
      return search(root->right, key);

   // Key is smaller than root's key
   return search(root->left, key);
}
```

# Binary Tree Traversal  Inorder, Preorder and Postorder

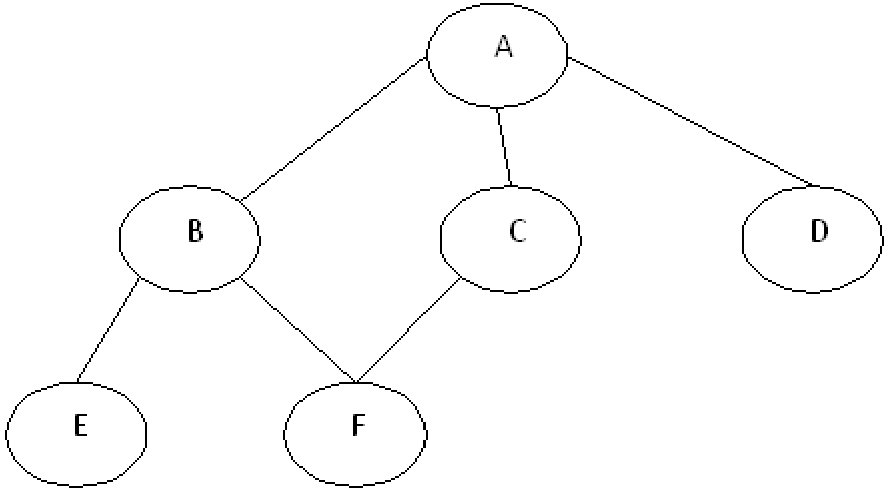


**Inorder Traversal:** 7 9 4 2 5 1 3 6 8
**Preorder Traversal:** 1 2 4 7 9 5 3 6 8
**Postorder Traversal:** 9 7 4 5 2 8 6 3 1

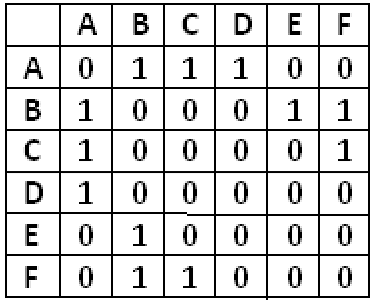
```
void inorder(node *temp)
{
    if(temp!=NULL)
    {
        inorder(temp->left);
        printf(" %d",temp->data);
        inorder(temp->right);
    }
}

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

```
void postorder(node *temp)
{
    if(temp!=NULL)
    {
        postorder(temp->left);
        postorder(temp->right);
        printf(" %d",temp->data);
    }
}
```

# **Program -6** Implement Graph using adjacency Matrix with BFS & DFS traversal



**Adjacency Matrix**



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 0 |
| B | 1 | 0 | 0 | 0 | 1 | 1 |
| C | 1 | 0 | 0 | 0 | 0 | 1 |
| D | 1 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 1 | 0 | 0 | 0 | 0 |
| F | 0 | 1 | 1 | 0 | 0 | 0 |

**Algorithmic Steps For DFS**

**Step 1**: Push the root node in the Stack.

**Step 2**: Loop until stack is empty.

**Step 3**: Peek the node of the stack.

**Step 4**: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.

**Step 5**: If the node does not have any unvisited child nodes, pop the node from the stack

**Algorithmic Steps for BFS**

**Step 1**: Push the root node in the Queue.

**Step 2**: Loop until the queue is empty.

**Step 3**: Remove the node from the Queue.

**Step 4**: If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

DFS.mp4 BFS.mp4

```
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);
}
  void  insert(int  x)
  {
                        p.rear++;
                        p.data[p.rear]=x;
  }
  int  dele()
  {
            int  x;
            x=p.data[p.front];
            if(p.rear==p.front)
            {
                        p.rear=-1;
                        p.front=0;
            }
            else
                        p.front++;
            return(x);
  }
```

```
void BFS(int v)
{
        int visited[max],i;
        void insert(int );
        p.rear=-1;
        p.front=0;
        for(i=0;i<n;i++)
         visited[i]=0;
        insert(v);
        printf("\n visit\n%d",v);
        visited[v]=1;
        while(!empty())
        {
        v=dele(); // visit and add adjacency vertices
                for(i=0;i<n;i++)
                        if(visited[i]==0 && G[v][i]!=0)
                        {
                                insert(i);
                                visited[i]=1;
                                printf("\n%d",i);
                        }
        }
}
```

# Data Structures

## Suggestions are Welcome!