

MA 511: Computer Programming  
**Lecture 13: Circular & Doubly Linked List**  
[http://www.iitg.ernet.in/psm/indexing\\_ma511/y10/index.html](http://www.iitg.ernet.in/psm/indexing_ma511/y10/index.html)

**Partha Sarathi Mandal**

[psm@iitg.ernet.ac.in](mailto:psm@iitg.ernet.ac.in)

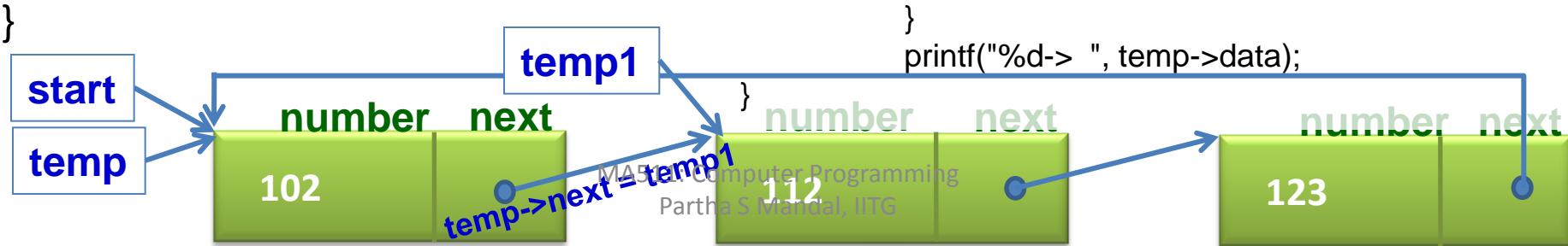
Dept. of Mathematics, IIT Guwahati

Semester 1, 2010-2011

# How to create a Circular linked list ?

```
struct list_of_no {  
    int data;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;  
void create(node *pt);  
void print(node *pt);  
main(){  
    node *start;  
    start = (node *) malloc(sizeof(node));  
    scanf("%d", &start->data);  
    create(start);  
    print(start);  
}
```

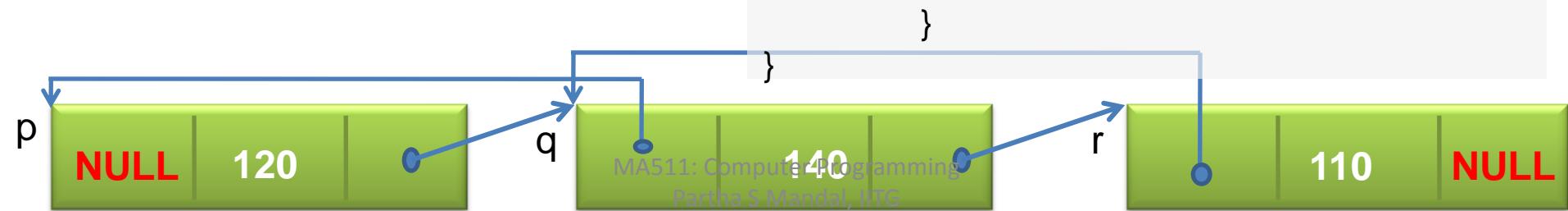
```
void create(node *head){  
    node *temp=head, *temp1;  
    char c_value;  
    while(1){  
        printf("Type 'Y' for continue Type 'N' for stop: ");  
        scanf(" %c", &c_value);  
        if(c_value=='N'){ temp->next = head; break; }  
        else{ temp1 = (node *) malloc(sizeof(node));  
              scanf("%d", &(temp1->data));  
              temp->next = temp1;  
              temp=temp->next; }  
    }  
void print(node *head){  
    node *temp=head;  
    printf("%d-> ", temp->data);  
    temp=temp->next;  
    while(!(temp==head)){  
        printf("%d-> ", temp->data);  
        temp=temp->next; }  
    printf("%d-> ", temp->data);  
}
```



# How to create a Doubly linked list?

```
struct list_of_no {  
    int data;  
    struct list_of_no *prev;  
    struct list_of_no *next;  
};  
typedef struct list_of_no node;
```

```
main(){  
    node *p, *q, *r;  
    p = (node *) malloc(sizeof(node));  
    q = (node *) malloc(sizeof(node));  
    r = (node *) malloc(sizeof(node));  
    p->data = 120;  
    q->data = 140;  
    p->data = 110;  
    p->prev = NULL;  
    p->next = q;  
    q->prev = p;  
    q->next = r;  
    r->prev = q;  
    r->next = NULL;  
    while(p!=NULL){  
        printf(" %d", p->data);  
        p=p->next;  
    }  
}
```



# Assignment

1. Write a C-Program for **inserting** a new node in a **doubly** linked list (i) before a target key, (ii) after a target key.
2. Write a C-Program for **deleting** a node from the **doubly** linked list (i) preceding to give key (ii) after a give key (iii) node containing the key value.
  - where target key, key value of the inserting node and deleting node should enter in from the console.

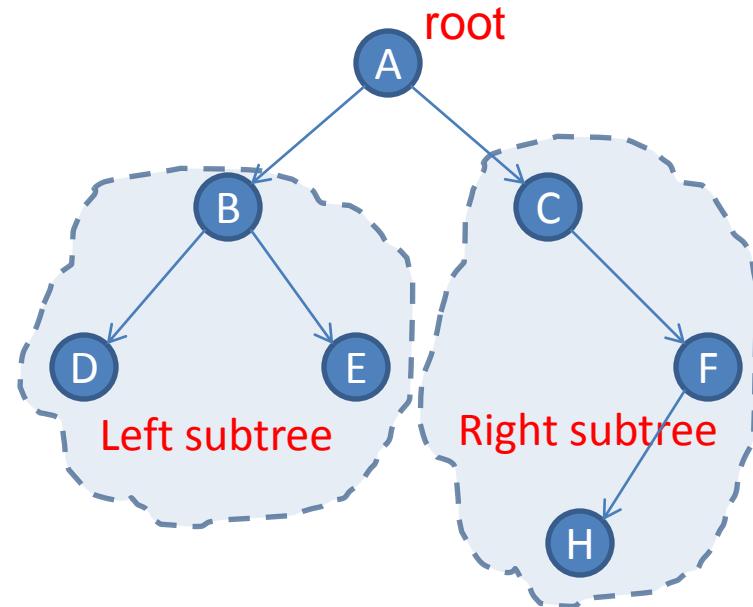
# Assignment

1. Write a C-Program for **inserting** a new node in a linked list (i) before a target key, (ii) after a target key.
2. Write a C-Program for **deleting** a node from the linked list (i) preceding to give key (ii) after a give key (iii) node containing the key value.
  - where target key, key value of the inserting node and deleting node should enter in from the console.

# Trees

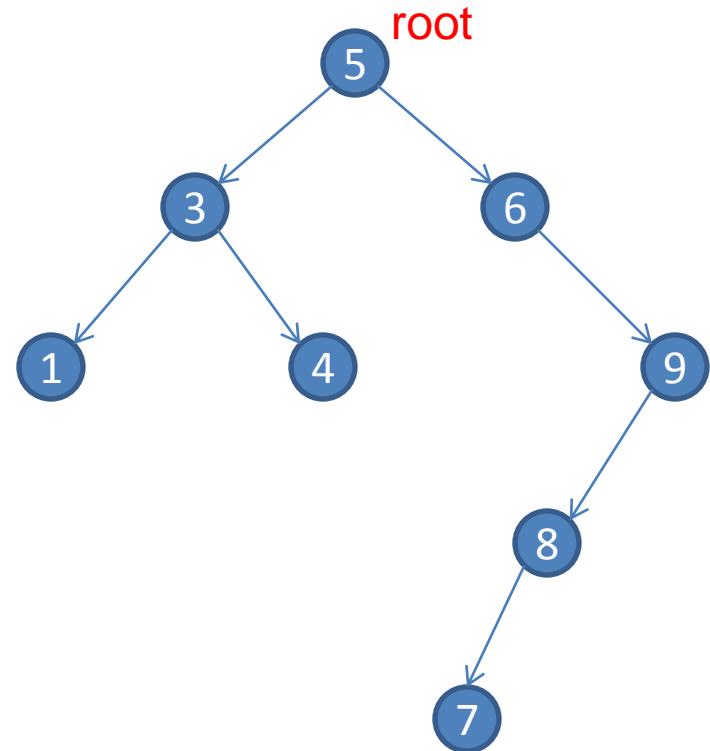
- **Binary Trees**

- A Binary tree is a finite set of elements that is either empty or is partitioned into at most three disjoint subsets.
- The first subset contains a single element called the **root** of the tree.
- The other two subsets are themselves binary trees, called the **left** and **right subtrees** of the original tree.



# Binary Search Trees

- Its a Binary Trees with following property
- All elements in the left subtree of a node **n** are less than the contents of **n**, and all elements in the right subtree of **n** are grater than or equal to the contents of **n**.

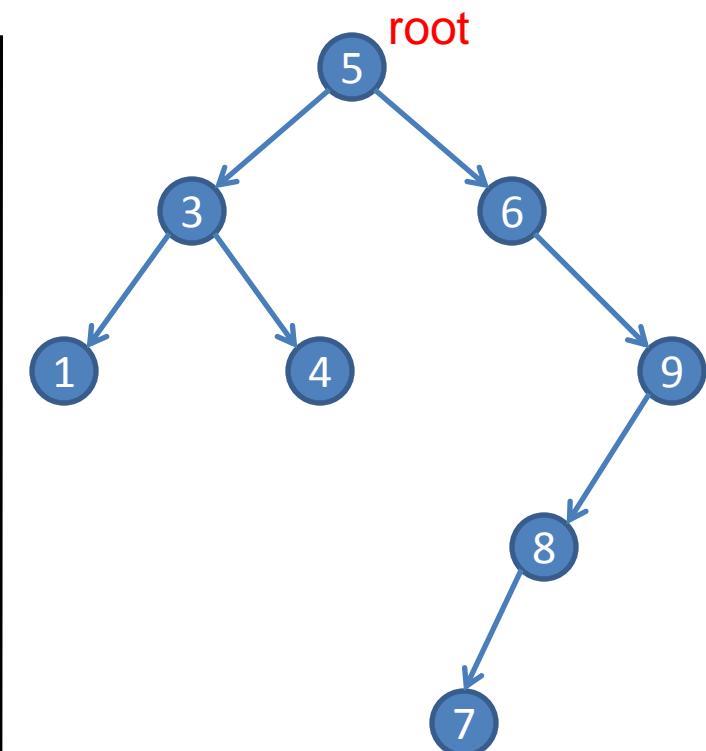
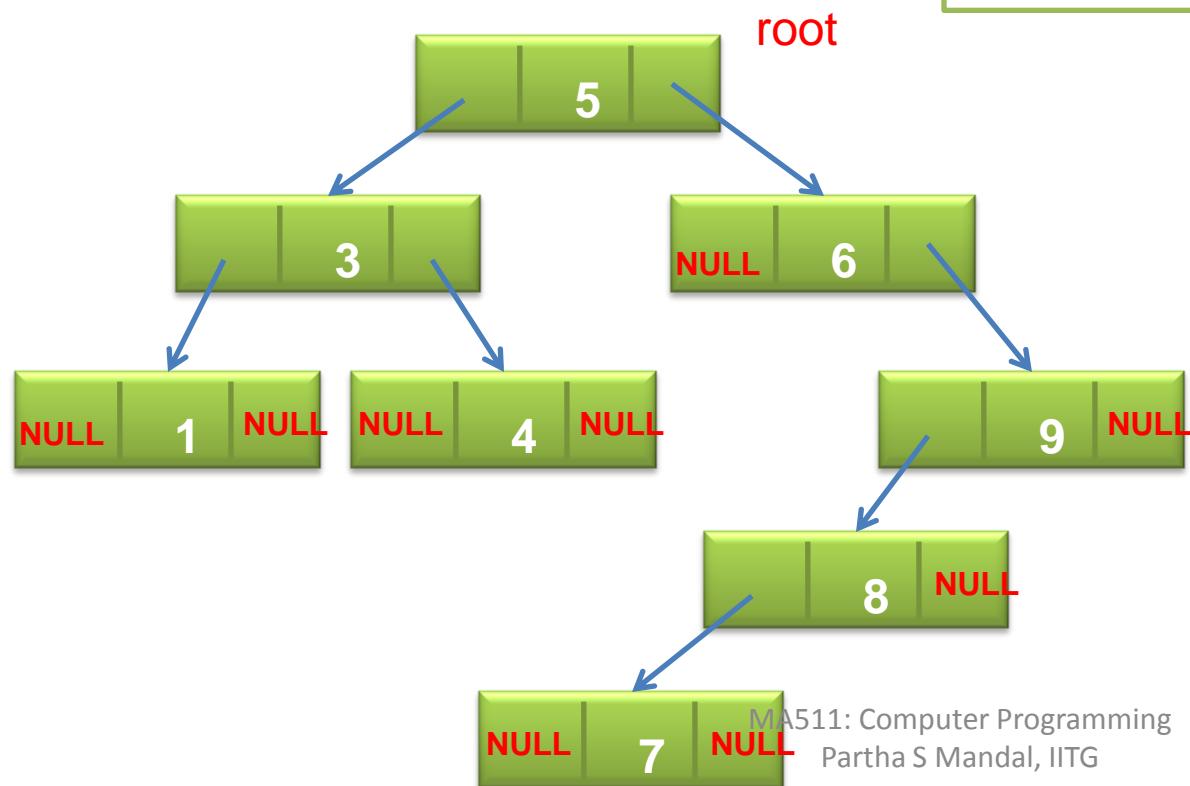


# Binary search tree using linked list

node



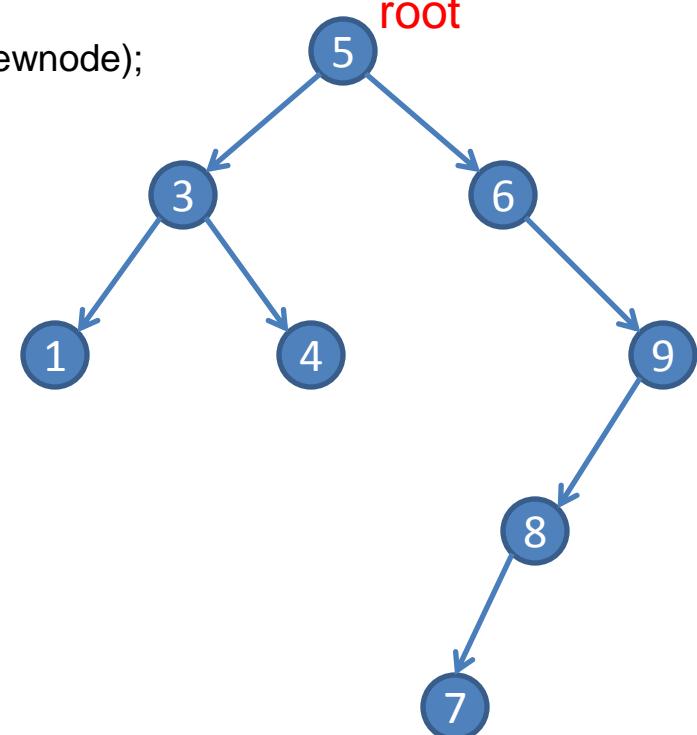
```
typedef struct BinaryTreeNode {  
    struct BinaryTreeNode *leftchild;  
    int data;  
    struct BinaryTreeNode *rightchild;  
} node;
```



# Code for Binary Search Trees

```
typedef struct BinaryTree {  
    struct BinaryTree *leftchild;  
    int data;  
    struct BinaryTree *rightchild;  
}node;  
node *root=NULL;  
main(){  
    node *newNode;  
    int i, n, key;  
    printf("Type no of nodes: ");  
    scanf("%d", &n);  
    for(i = 0; i < n; i++){  
        printf("Type %dth data : ", i);  
        scanf("%d", &key);  
        newNode = createNode(key);  
        add_node(newNode);  
    }  
    print_inorder(root);  
}
```

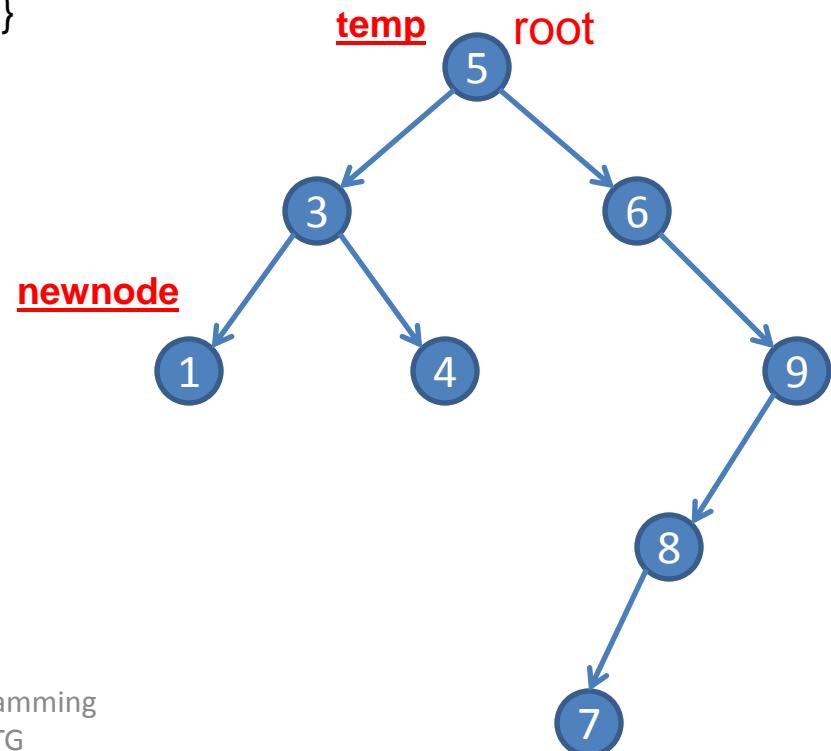
```
node *createNode(int value){  
    node *newnode;  
    newnode = (node *)malloc(sizeof(node));  
    if (newnode) {  
        newnode->data = value;  
        newnode->leftchild = NULL;  
        newnode->rightchild = NULL;  
    }  
    return(newnode);  
}
```



# Code for Binary Search Trees

```
void add_node(node *newnode){  
    node *temp;  
    if (!root) root = newnode;  
    else {  
        temp = root;  
        while(1){  
            if(newnode->data < temp->data){  
                if(!temp->leftchild){  
                    temp->leftchild = newnode; break;  
                }  
                else temp = temp->leftchild;  
            }  
            else{  
                if(!temp->rightchild){  
                    temp->rightchild = newnode; break;  
                }  
                else temp = temp->rightchild;  
            }  
        }  
    }  
}
```

```
void print_inorder(node *node) {  
    if(node == NULL) return;  
  
    print_inorder(node->leftchild);  
    printf("%d ", node->data);  
    print_inorder(node->rightchild);  
}
```



# Searching target in a given Tree

```
/* Given a binary tree, return 1 if a node with the
target data is found in the tree otherwise return 0.
Recurrs down the tree, chooses the left or right
branch by comparing the target to each node.*/

int search(struct node* temp, int target) {
    if (temp == NULL) return(0);           //not found
    else {
        if (target == temp->data) return(1); //found here
        else { //recur down to the correct subtree
            if (target < temp->data)
                return(search(temp->left, target));
            else
                return(search(temp->right, target));
        }
    }
}
```

```
main{
    :
    :
    printf("Type Target: ");
    scanf("%d", &target);
    j = search(root, target);
    if(!j) printf("Target NOT in tree");
    else printf("Target IN the Tree");
    :
    :
}
```

# Traversal of a Binary Tree

- Preorder
  - Visit the root
  - Traverse the left subtree in preorder
  - Traverse the right subtree in preorder

Ex. 5 3 1 4 6 9 8 7
- Inorder
  - Traverse the left subtree in inorder
  - Visit the root
  - Traverse the right subtree in inorder

Ex. 1 3 4 5 6 7 8 9
- Postorder
  - Traverse the left subtree in postorder
  - Traverse the right subtree in postorder
  - Visit the root

Ex. 1 4 3 7 8 9 6 5

