```
struct TreeNode{
    int data;
    TreeNode* left;
    TreeNode* right; };
TreeNode* root;
int isLeaf(TreeNode* ptrNode){
    if(ptrNode->left == NULL & ptrNode->right ==NULL){
        return 1;}
    else {
        return 0;}
int size(TreeNode* node){
    if(node == NULL )
        return 0;
    else{
        return 1 + size(node->left) +
                size(node->right);}
}
void print(TreeNode* node){
    if(node != NULL){
        cout << node->data;
        print(node->left);
        print(node->right);}
}
TreeNode* addNode(int treeData, TreeNode* node){
    if(node == NULL){
        node = (TreeNode*)malloc(sizeof(struct TreeNode));
        node->data = treeData;}
    else if(treeData > node->data){//go right
        addNode(treeData, node->right);}
    else {//go left
        addNode(treeData, node->left);}
    return node;}
void addnode (int dat, TreeNode* part){
    if (data < (part -> data)) {
        if(part -> left == NULL){
            part -> left = (TreeNode*)malloc(sizeof(struct TreeNode));
            part -> left -> data = data;}
        else{
            addnode(dat, part -> left);}
    }
    else{
        if(part -> right == NULL){
            part -> right = (TreeNode*)malloc(sizeof(struct TreeNode));
            part -> right -> data = dat;
        }
       else{
            addnode(dat, part -> right);}
    }
}
int main(int argc, char** argv) {
    root = (TreeNode*)malloc(sizeof(struct TreeNode));
    root->data = 5;
    addnode(3, root);
    print(root);
    return (EXIT_SUCCESS);}
struct node_double{
    int data;
    node_double* next;
    node_double* prev;
};
------------------------------
node_double* head;
node_double* tail;
void addNode(int x){
    if(head==NULL){
        head=(node_double*)malloc(sizeof(struct node_double));
        head->data = x;
        head->next = NULL;
        head->prev = NULL;
        tail=head;}
    else{
        tail->next = (node_double*)malloc(sizeof(struct node_double));
        tail->next->data = x;
        tail->next->prev = tail;
        tail->next->next=NULL;
        tail = tail->next;}
};
void printList(node_double* headOfList){
    node_double* tmp;
    tmp = headOfList;
    while(tmp->next !=headOfList){
        cout << "Values: " << tmp-> data << "\n";
        tmp = tmp->next;
    }
};


void printReverseList(node_double* tailOfList){
    node_double* tmp;
    tmp = tailOfList;

    while(tmp->prev !=tailOfList){

        cout << "Values: " << tmp-> data << "\n";

        tmp = tmp->prev;

    }

}

void addNode(node* newn){

    newn->next = head; //The new node points to head (memory address) that is
the reference to the first value

    head = newn; //Pointer head points to the new node

}

void deleteNode(){

    node* prev = head; //Node previous to the actual node

    node* aux = head; //Auxiliar node equal to the head

    while(aux-> next != NULL){ //Checks if the next value of the node is null

        prev=aux;               //First prev equals to aux, so we don't lose the reference
of aux

        aux = aux->next;        //We move to the next node

    }

    prev->next = NULL;          //We lose the reference of the last element
"eliminate"

    delete aux; // We liberate that space of memory

}
```