Q1. Create a binary tree. User shall input the number of nodes. Program should have a separate insert function. Implement in order, pre order and post order traversal recursively. Display the nodes travelled in each case.

Ans.

#include <stdio.h>

#include <stdlib.h>

struct Node{

    int val;

    struct Node\* left;

    struct Node\* right;

};

struct Node\* create\_node(int val){

    struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

    node->left = NULL;

    node->right = NULL;

    node->val = val;

    return node;

}

//Queue is necessary to make the binary tree, so we make a circular queue structure.

struct QNode {

    struct Node\* node;

    struct QNode\* next;

};

struct Queue {

    struct QNode \*front, \*rear;

};

struct QNode\* newNode(struct Node\* cur\_node)

{

    struct QNode\* temp = (struct QNode\*)malloc(sizeof(struct QNode));

    temp->node = cur\_node;

    temp->next = NULL;

    return temp;

}

struct Queue\* createQueue()

{

    struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));

    q->front = q->rear = NULL;

    return q;

}

void enQueue(struct Queue\* q, struct Node\* cur\_node)

{

     struct QNode \*node = newNode(cur\_node);

     if(q->front == NULL && q->rear == NULL){

         node->next = node;

         q->front = q->rear = node;

     }

     else{

         q->rear->next = node;

         node->next = q->front;

         q->rear = node;

     }

}

struct Node\* deQueue(struct Queue\* q)

{

    if(q->front == NULL && q->rear == NULL){

        return NULL;

    }

    else if(q->front == q->rear){

        struct Node\* r\_node = q->front->node;

        q->front->next = q->rear->next = NULL;

        free(q->front);

        q->front = q->rear = NULL;

        return r\_node;

    }

    else{

        struct Node\* r\_node = q->front->node;

        struct QNode \*node = q->front;

        q->front = q->front->next;

        q->rear->next = q->front;

        node->next = NULL;

        free(node);

        return r\_node;

    }

}

//Queue structure ends here

void inorder(struct Node\* root){

    if(root != NULL){

        inorder(root->left);

        printf("%d ", root->val);

        inorder(root->right);

    }

}

void preorder(struct Node\* root){

    if(root != NULL){

        printf("%d ", root->val);

        preorder(root->left);

        preorder(root->right);

    }

}

void postorder(struct Node\* root){

    if(root != NULL){

        postorder(root->left);

        postorder(root->right);

        printf("%d ", root->val);

    }

}

struct Node\* insert(int nodes){

    struct Queue\* q = createQueue();

    int val;

    printf("Enter values for nodes: \n");

    scanf("%d", &val);

    struct Node\* cur\_node = create\_node(val);

    enQueue(q, cur\_node);

    struct Node\* root = cur\_node;

    nodes--;

    while(nodes>0){

        cur\_node = deQueue(q);

        scanf("%d", &val);

        cur\_node->left = create\_node(val);

        nodes--;

        enQueue(q, cur\_node->left);

        if(nodes == 0){

            break;

        }

        scanf("%d", &val);

        cur\_node->right = create\_node(val);

        nodes--;

        enQueue(q, cur\_node->right);

    }

    return root;

}

int main()

{

    int nodes;

    printf("Enter number of nodes: ");

    scanf("%d", &nodes);

    if(nodes>0){

        struct Node\* root = insert(nodes);

        printf("\nInorder Traversal: ");

        inorder(root);

        printf("\nPreorder Traversal: ");

        preorder(root);

        printf("\nPostorder Traversal: ");

        postorder(root);

    }

    else{

        printf("Number of nodes must be greater than 0");

    }

    return EXIT\_SUCCESS;

}

Output



