S.No: 12	Exp. Name: ***Binary tree traversals using linked list - Recursive***	**Date:2023-11-16**

Aim:

Write a C program to implement Binary tree traversals using Linked list.
You have to complete the function `inorder`, `preorder` and `postorder` in `Traversarls.c` where parameters passed are the root reference's of the binary tree **T1**.

Source Code:

TreeMain.c

```
// Program for linked implementation of complete binary tree
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include<stdbool.h>
#include"TreeStructure.c"
#include"Traversals.c"


// For Queue Size
#define SIZE 100000



// A utility function to create a new Queue
struct Queue* createQueue(int size)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof( struct Queue ));

    queue->front = queue->rear = -1;
    queue->size = size;

    queue->array = (struct node**) malloc
                   (queue->size * sizeof( struct node* ));

    int i;
    for (i = 0; i < size; ++i)
        queue->array[i] = NULL;

    return queue;
}

// Standard Queue Functions
int isEmpty(struct Queue* queue)
{
    return queue->front == -1;
}

int isFull(struct Queue* queue)
{ return queue->rear == queue->size - 1;

}
```

```
int hasOnlyOneItem(struct Queue* queue)
{  return queue->front == queue->rear;
}

void Enqueue(struct node *root, struct Queue* queue)
{
    if (isFull(queue))
        return;
    queue->array[++queue->rear] = root;
    if (isEmpty(queue))
        ++queue->front;
}
struct node* Dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
    struct node* temp = queue->array[queue->front];

    if (hasOnlyOneItem(queue))
        queue->front = queue->rear = -1;
    else
        ++queue->front;
    return temp;
}

struct node* getFront(struct Queue* queue)
{  return queue->array[queue->front]; }

// A utility function to check if a tree node
// has both left and right children
int hasBothChild(struct node* temp)
{
    return temp && temp->left && temp->right;
}

// Function to insert a new node in complete binary tree
void insert(struct node **root, int data, struct Queue* queue)
{
    // Create a new node for given data
    struct node *temp = newNode(data);
    // If the tree is empty, initialize the root with new node.
    if (!*root)
        *root = temp;
    else
    {
        // get the front node of the queue.
        struct node* front = getFront(queue);

        // If the left child of this front node doesn’t exist, set the
        // left child as the new node
        if (!front->left)
        {
                front->left = NULL;
            front->left = temp;
        }
        // If the right child of this front node doesn’t exist, set the
        // right child as the new node
        else if (!front->right)
        {
                front->right = NULL;
            front->right = temp;
        }
        // If the front node has both the left child and right child,
        // Dequeue() it.
        if (hasBothChild(front))
        {
                Dequeue(queue);
        }
    }
    // Enqueue() the new node for later insertions
            Enqueue(temp, queue);
}

// Standard level order traversal to test above function
void levelOrder(struct node* root)
{
    struct Queue* queue = createQueue(SIZE);
    Enqueue(root, queue);
    while (!isEmpty(queue))
    {
        struct node* temp = Dequeue(queue);
        printf("%d ", temp->data);
        if (temp->left)
            Enqueue(temp->left, queue);
        if (temp->right)
            Enqueue(temp->right, queue);
    }
}

// Driver program to test above functions
int main()
{
    struct node* T1 = NULL;
    struct Queue* queue1 = createQueue(SIZE);
    int ele;

    while(1){
       printf("Enter value : ");
       scanf("%d",&ele);
    if(ele==-1){
        break;
    }
    else{
           insert(&T1, ele, queue1);
       }
    }
    inorder(T1);
    printf("\n");
    preorder(T1);
    printf("\n");
    postorder(T1);
}
```

TreeStructure.c

```c
// A tree node
struct node
{
    int data;
    struct node *right,*left;
    struct node *root;
};

// A queue node
struct Queue
{
    int front, rear;
    int size;
    struct node* *array;
};

// A utility function to create a new tree node
struct node* newNode(int data)
{
    struct node* temp = (struct node*) malloc(sizeof( struct node ));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```

Traversals.c

```c
void inorder( struct node *ptr)
{

     if (ptr)
    {

        inorder(ptr->left);
        printf("%d ", ptr->data);
        inorder(ptr->right);
    }
}

void preorder( struct node *ptr){
    if (ptr)
    {
        printf("%d ", ptr->data);
        preorder(ptr->left);
        preorder(ptr->right);
    }
}

void postorder(struct node *ptr)
{
   if (ptr)
    {
        postorder(ptr->left);
        postorder(ptr->right);
         printf("%d ", ptr->data);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
`Enter value :  45`
`Enter value :  89`
`Enter value :  56`
`Enter value :  12`
`Enter value :  45`
`Enter value :  26`
`Enter value :  78`
`Enter value :  45`
`Enter value :  -1`
`45 12 89 45 45 26 56 78`
`45 89 12 45 45 56 26 78`
`45 12 45 89 26 78 56 45`

Test Case - 2
User Output
`Enter value :  15`
`Enter value :  12`
`Enter value :  14`
`Enter value :  18`
`Enter value :  -1`
`18 12 15 14`
`15 12 18 14`
`18 12 14 15`