Linked complete binary tree & its creation

A complete binary tree is a binary tree where each level 'l' except the last has 2^l nodes and the nodes at the last level are all left aligned. Complete binary trees are mainly used in heap based data structures.
The nodes in the complete binary tree are inserted from left to right in one level at a time. If a level is full, the node is inserted in a new level.

Below are some of the complete binary trees.

```
       1
      / \
     2   3

        1
       / \
      2   3
     / \  /
    4  5 6
```

Below binary trees are not complete:

```
     1
    / \
   2   3
  /    /
  4   5

       1
      / \
     2   3
    / \  /
   4  5 6
  /
 7
```

Complete binary trees are generally represented using arrays. The array representation is better because it doesn't contain any empty slot. Given parent index i, its left child is given by 2 * i + 1 and its right child is given by 2 * i + 2. So no extra space is wasted and space to store left and right pointers is saved. However, it may be an interesting programming question to create a Complete Binary Tree using linked representation. Here Linked mean a non-array representation where left and right pointers(or references) are used to refer left and right children respectively. How to write an insert function that always adds a new node in the last level and at the leftmost available position?
To create a linked complete binary tree, we need to keep track of the nodes in a level order fashion such that the next node to be inserted lies in the leftmost position. A queue data structure can be used to keep track of the inserted nodes.

Following are steps to insert a new node in Complete Binary Tree.
1. If the tree is empty, initialize the root with new node.

2. Else, get the front node of the queue.
…….If the left child of this front node doesn't exist, set the left child as the new node.
…….else if the right child of this front node doesn't exist, set the right child as the new node.

3. If the front node has both the left child and right child, Dequeue() it.

4. Enqueue() the new node.

Below is the implementation:

```
// Program for linked implementation of complete binary tree
#include <stdio.h>
#include <stdlib.h>

// For Queue Size
```

```c
#define SIZE 50

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

// 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;
}

// 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 = 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 = 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* root = NULL;
    struct Queue* queue = createQueue(SIZE);
    int i;

    for(i = 1; i <= 12; ++i)
        insert(&root, i, queue);

    levelOrder(root);

    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```