Level Order Traversal

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 500

struct node
{
	int data;
	struct node *left;
	struct node *right;
};

struct node *newNode(int data)
{
	struct node *temp = (struct node *)malloc(sizeof(struct node));
	temp->data = data;
	temp->left = temp->right = NULL;
	return temp;
}

struct node **createQueue(int *front, int *rear)
{
	struct node **queue = (struct node **)malloc(sizeof(struct node *)*MAX);
	*front = *rear = 0;
	return queue;
}

void enqueue(struct node **queue, int *rear, struct node *newNode)
{
	queue[(*rear)++] = newNode;
}

struct node *dequeue(struct node **queue, int *front)
{
	return queue[(*front)++];
}

int isQueueEmpty(int *rear, int *front)
{
	return ((*rear) == (*front));
}

void getLevelOrder(struct node *root)
{
	int front, rear;
	if(root)
	{
		struct node **queue = createQueue(&front, &rear);
		struct node *temp;
		enqueue(queue, &rear, root);
		while(!isQueueEmpty(&rear, &front))
```

```
                {
                        temp = dequeue(queue, &front);
                        printf("%d\t", temp->data);
                        if(temp->left)
                                enqueue(queue, &rear, temp->left);
                        if(temp->right)
                                enqueue(queue, &rear, temp->right);
                }
        }
}

int main()
{
        struct node *root=NULL;
        root = newNode(10);
        root->left = newNode(20);
        root->right = newNode(30);
        root->left->left = newNode(40);
        root->right->left = newNode(50);
        root->right->right = newNode(60);
        getLevelOrder(root);
        return 0;
}
```

Time complexity: O(n)
Space complexity: O(n)