Level–Order Traversal of a Binary Search Tree or Breadth First Search on Binary Search Tree

Dr. Rahul Das Gupta

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 10

typedef struct bs_tree
{
    int data;
    struct bs_tree *left, *right;
}BS_TREE;

typedef struct queue
{
   BS_TREE *elements[MAX_QUEUE_SIZE];
   int front, rear;
}QUEUE;
```

```
/* ============================
Here, element is an array of pointers.
Data type of element : BS_TREE **
============================*/

void initialiseQueue (QUEUE *);
void addQueue (BS_TREE *, QUEUE *);
BS_TREE * deleteQueue (QUEUE *);
int isEmptyQueue (QUEUE );
void initialise_tree (BS_TREE **);
void rec_insert (BS_TREE **, int  );
void level_order_Trversal (BS_TREE *);



void initialiseQueue (QUEUE *aq)
{
    aq->front =-1;
    aq->rear =-1;
}

void addQueue (BS_TREE * at, QUEUE *aq)
{
```

```
 /*There is no space in Queue for the new Item.
Item can not be added in the Queue.*/
   if (aq->rear = = MAX_QUEUE_SIZE-1)
            exit(0); /* Unsuccessful Addition*/
 /* Spaces are available in Queue for the new
Item. */
   else
              aq->elements [++(aq->rear)]=at;
/* Successful Addition*/
}

BS_TREE * deleteQueue (QUEUE *aq)
{
    BS_TREE *p;
   /* Non-empty Queue*/
   if (aq->front != aq->rear)
   {
        p= aq->elements [++(aq->front)];
        if (aq->front == aq->rear)
        /* Queue is empty*/
                  initialiseQueue (aq);
        return p; /* Successful retrieval of data*/
   }
   else /* Empty Queue*/
```

```
            return NULL; /* Empty Queue*/
    }

int  isEmptyQueue (QUEUE q)
{
    return (q.front = = -1);
}

 void initialise_tree (BS_TREE **aar)
{
        *aar = NULL;
}

void rec_insert (BS_TREE **aar, int n)
{
    if (*aar == NULL)
    {
      *aar=(BS_TREE *)malloc(sizeof(BS_TREE));
      (*aar)->data=n;
      (*aar)->left=(*aar)->right=NULL;
    }
    else if (n<(*aar)->data)
          rec_insert(&((*aar)->left), n);
    else
         rec_insert(&((*aar)->right), n);
```

```
}



void level_order_Trversal (BS_TREE *ar)
{
      QUEUE q;
      BS_TREE *ptr;
      /*Initialisation of Queue*/
      initialiseQueue (&q);
     /*Insert the Root of the tree in the Queue.*/
      addQueue (ar, &q);
     /*Continue untill Queue is empty.*/
      while (! isEmpty(q))
     {
        /*Remove the first address from the
Queue to a ptr.*/
           ptr = deleteQueue (&q);
 /*Insert the left child of a ptr in the Queue.*/
           if ( ptr->left != NULL )
                addQueue (ptr->left, &q);
/* Insert the right child of a ptr in the Queue.*/
           if ( ptr->right != NULL )
```

```
            addQueue (ptr->right, &q);
        printf("\t %d", ptr->data);
    }
}

void main( )
{
    BS_TREE *t;
    initialise_tree(&t);

    rec_insert (&t, 8);
    rec_insert (&t, 10);
    rec_insert (&t, 9);
    rec_insert (&t, 12);
    rec_insert (&t, 15);
    rec_insert (&t, 20);
    rec_insert (&t, 25);
    rec_insert (&t, 30);

    level_order_Trversal (t);
}
```