```c

/**
 * Recursive traversal for Threaded Binary Tree
 * Author: Mithusayel Murmu
 */

#include <stdio.h>
#include <stdlib.h>

/* Handy typedefs */
typedef struct _TBT TBT;
typedef struct _TBTNode TBTNode;
typedef enum { FALSE, TRUE } BOOL;

/* Threaded Binary Tree and Node definitions */
struct _TBTNode {
    int data;
    TBTNode *left, *right;
    BOOL hasLThread, hasRThread;
};
struct _TBT { TBTNode *root; size_t size; };

/* Creates and returns a pointer to an empty TBT */
static TBT * tbt_create_tree() {
    TBT *tree = (TBT *) malloc(sizeof(TBT));
    tree->root = NULL; tree->size = 0;
    return tree;
}

/* Creates and returns a pointer to an empty TBTNode */
static TBTNode * tbt_create_node(int data,
                                 TBTNode *lt, TBTNode *rt, BOOL lTh, BOOL rTh) {
    TBTNode *node = (TBTNode *) malloc(sizeof(TBTNode));
    node->data = data;
    node->left = lt; node->right = rt;
    node->hasLThread = lTh; node->hasRThread = rTh;
    return node;
}

static void tbt_insert_node(TBTNode **node, int data, TBTNode *lt, TBTNode *rt) {
    // Reached a leaf
    if (*node == NULL) {
        *node = tbt_create_node(data, lt, rt, TRUE, TRUE);
        return;
    }

    TBTNode *nd = *node;
    // Redirect left from non-leaf node
    if (data < nd->data) {
        // Reset current node's left thread state
        if (nd->hasLThread) { nd->hasLThread = FALSE; nd->left = NULL; }
        tbt_insert_node(&(nd->left), data, lt, nd);
    } else {  // Redirect right from non-leaf node
        // Reset current node's right thread state
        if (nd->hasRThread) { nd->hasRThread = FALSE; nd->right = NULL; }
        tbt_insert_node(&(nd->right), data, nd, rt);
    }
}

void tbt_insert(TBT *tree, int data) {
    tbt_insert_node(&(tree->root), data, NULL, NULL);
    tree->size++;
}

#define SEEK_LEFT(node) \
    while ((node)->left && !(node)->hasLThread) (node) = (node)->left;

void tbt_traverse(TBT *tree, void (*callback)(int)) {
    if (tree == NULL || tree->size == 0)
        return;

    BOOL isRP;
    TBTNode *node = tree->root;
    SEEK_LEFT(node);

    while (node) {
        callback(node->data);
        isRP = !node->hasRThread;
        node = node->right;
```

```c

        if (node && isRP) SEEK_LEFT(node);
    }
}

static void print_utility(int n) { printf("%d ", n); }

/** Driver function */
int main(int argc, const char *argv[]) {
    int N, num;
    TBT *tree = tbt_create_tree();

    printf("Number of elements to be inserted: ");
    scanf("%d", &N);
    printf("Enter %d space separated integers: ", N);

    while (N--) {
        scanf("%d", &num);
        tbt_insert(tree, num);
    }

    printf("\nPrinting while traversal:\n");
    tbt_traverse(tree, print_utility);
    printf("\n");

    return 0;
}

```