BINARY TREE

Function BUILDTREE(ROOT,NDX)

Given an array, LIST representing a level-traversed tree, this unction builds a tree rooted at LIST[0]. The elements, LIST[NDX] indicates the node value for tree nodes. If the node is not present, it is indicated by a special value NODATA [say -1]. The last element of the array is stored as MAXVALUE to indicate end of array. TEMP is the local tree pointer.

```
1.  Initailize TEMP.
        TEMP = NULL

2.  Recursively Build the tree
        If LIST[NDX] <> NODATA
          TEMP <== NODE
          LCHILD(TEMP) = Call BUILDTREE(ROOT,Call GETLCHILD(ROOT,NDX))
          DATA(TEMP) = LIST[NDX]
          RCHILD(TEMP) = Call BUILDTREE(ROOT,Call GETRCHILD(ROOT,NDX))

3.  Return the tree
        Return TEMP
```

Function GETLCHILD(FIRST, INDEX)

Given an array, LIST representing a level-traversed tree, this function returns the index of left child of the node pointed by INDEX. LIST.Length denotes the length of the list, LIST whose last element is set to MAXVALUE to indicate end of array.

```
1.  Check if the indicated node exits
        If (2*NDX+1) > LIST.Length-1
            Return(LIST.Length-1)

2.  Return the left child index
        Return (2*NDX+1)
```

Function GETRCHILD(FIRST, INDEX)

Given an array, LIST representing a level-traversed tree, this function returns the index of right child of the node pointed by NDX. LIST.Length denotes the length of the list, LIST whose last element is set to MAXVALUE to indicate end of array.

```
1.  Check if the indicated node exits
        If (2*INDEX+2) > LIST.Length-1
            Return(LIST.Length-1)

2.  Return the right child index
        Return (2*INDEX+2)
```

Function INORDER(ROOT)

Given a rooted binary tree pointed by ROOT, this function prints the indorder traversal of the tree [LCHILD--ROOT--RCHILD] recursively.

```
1.  Recursion??
    If ROOT <> NULL
        //Traverse the Left Sub-Tree
            Call INORDER(LCHILD(ROOT)

        //Process the ROOT
            Write(DATA(ROOT))

        //Traverse the Right Sub-Tree
            Call INORDER(RCHILD(ROOT)
```

Function PREORDER(ROOT)

Given a rooted binary tree pointed by ROOT, this function prints the predorder traversal of the tree [ROOT--LCHILD--RCHILD] recursively.

```
1.  Recursion??
    If ROOT <> NULL
        //Process the ROOT
            Write(DATA(ROOT))

        //Traverse the Left Sub-Tree
            Call PREORDER(LCHILD(ROOT)

        //Traverse the Right Sub-Tree
            Call PREORDER(RCHILD(ROOT)
```

Function POSTORDER(ROOT)

Given a rooted binary tree pointed by ROOT, this function prints the postorder traversal of the tree [LCHILD--RCHILD--ROOT] recursively.

```
1.  Recursion??
    If ROOT <> NULL
        //Traverse the Left Sub-Tree
            Call PREORDER(LCHILD(ROOT)

        //Traverse the Right Sub-Tree
            Call PREORDER(RCHILD(ROOT)

        //Process the ROOT
            Write(DATA(ROOT))
```

```c
*/
    Creation of a binary tree from its array representation. NDT
    indicates the absence of a node.

    Author    : Prof. D. A. Borikar, RCOEM
    Date      : 05-09-2017
*/

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

#define MX  50
#define NDT -1
#define MXVAL 999

/**Tree Node Definition */
    struct treeNode {
        int data;
        struct treeNode *lchild;
        struct treeNode *rchild;
    };

    typedef struct treeNode tNode;
    typedef tNode* tree;

/**Function Declarations */
    tree buildTree(int *, int, int);
    void inOrderT(tree);
    void postOrderT(tree);
    void preOrderT(tree);
    int getLeftChild(int *, int, int);
    int getRightChild(int *, int, int);

/**The Driver Function */
int main(){
    int len, list[MX] = {9,6,3,5,NDT,4,1,NDT,MXVAL};
    tree root;

    len = lengthList(list);
    printf("\nLength:= %d\n", len);

    root = buildTree(list, len, 0);
    printf("\nInorder Traversal...  : ");
    inOrderT(root);
    printf("\n");

    printf("\nPreorder Traversal... : ");
    preOrderT(root);
    printf("\n");

    printf("\nPostorder Traversal...: ");
    postOrderT(root);
    printf("\n\n");

    return 0;
}
```

```
/**Function Definitions */
tree buildTree(int list[], int len, int ndx){
    tree temp = NULL;

    if(list[ndx] != NDT){
        temp = (tree) calloc(1, sizeof(tree));
        temp->lchild = buildTree(list,len, getLChild(list, len, ndx));
        temp->data = list[ndx];
        temp->rchild = buildTree(list,len, getRChild(list, len, ndx));
    }
    return temp;
}

void preOrderT(tree root){
    if(root != NULL) {
        printf("%4d", root->data);
        preOrderT(root->lchild);
        preOrderT(root->rchild);
    }
}

void inOrderT(tree root){
    if(root != NULL) {
        inOrderT(root->lchild);
        printf("%4d", root->data);
        inOrderT(root->rchild);
    }
}

void postOrderT(tree root){
    if(root != NULL) {
        postOrderT(root->lchild);
        postOrderT(root->rchild);
        printf("%4d", root->data);
    }
}

int getLChild(int list[], int len, int ndx){
    int ele;
    if((2*ndx+1) > len)
        return len;
    return (2*ndx+1);
}

int getRChild(int list[], int len, int ndx){
    int ele;
    if((2*ndx+2) > len)
        return len;
    return (2*ndx+2);
}

int lengthList(int list[]){
    int i = 0;
    while(list[i] != MXVAL)
        i = i + 1;
    return i-1;
}
```