```
// ***
// *** You MUST modify this file
// ***

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tree.h"

// DO NOT MODIFY FROM HERE --->>>
void deleteTreeNode(TreeNode * tr)
{
  if (tr == NULL)
    {
      return;
    }
  deleteTreeNode (tr -> left);
  deleteTreeNode (tr -> right);
  free (tr);
}

void freeTree(Tree * tr)
{
  if (tr == NULL)
    {
      // nothing to delete
      return;
    }
  deleteTreeNode (tr -> root);
  free (tr);
}

static void preOrderNode(TreeNode * tn, FILE * fptr)
{
  if (tn == NULL)
    {
      return;
    }
  fprintf(fptr, "%d\n", tn -> value);
  preOrderNode(tn -> left, fptr);
  preOrderNode(tn -> right, fptr);
}

void preOrder(Tree * tr, char * filename)
{
  if (tr == NULL)
    {
      return;
    }
  FILE * fptr = fopen(filename, "w");
```

```
  preOrderNode(tr -> root, fptr);
  fclose (fptr);
}
// <<<--- UNTIL HERE

// ***
// *** You MUST modify the follow function
// ***

#ifdef TEST_BUILDTREE
// Consider the algorithm posted on
// https://www.geeksforgeeks.org/construct-a-binary-tree-from-
postorder-and-inorder/
int findIndex(int arr [], int treeRoot, int size)
{
 int index;
 for(index = 0; index < size; index++)
 {
  if(arr[index] == treeRoot)
  {
   break;
  }
 }
 return index;
}

TreeNode * createNode(int num)
{
 TreeNode * node = malloc(sizeof(TreeNode));
 node -> value = num;
 node -> left = NULL;
 node -> right = NULL;
 return node;
}

TreeNode * recursiveBuild(int inArray[], int start, int end, int
postArray[], int * postIndex)
{
 if(start > end)
 {
  return NULL;
 }

 TreeNode * newNode = createNode(postArray[*postIndex]); //creates
root
 (*postIndex) --; //moves root

 if(start == end)
 {
  return newNode;
```

```
 }

 int index = findIndex(inArray, newNode -> value, end);
 newNode -> right = recursiveBuild(inArray, index + 1, end, postArray,
postIndex);
 newNode -> left = recursiveBuild(inArray, start, index - 1,
postArray, postIndex);
 return newNode;
}

/*TreeNode * createNode(int num)
{
 TreeNode * node = malloc(sizeof(TreeNode);
 node -> value = num;
 node -> left = NULL;
 node -> right = NULL;
 return node;
}*/

Tree * buildTree(int * inArray, int * postArray, int size)
{
 int treeRoot = postArray[size - 1];

 Tree * t = malloc(sizeof(Tree));
 t -> root = createNode(treeRoot);

 int ind = findIndex(inArray, treeRoot, size);

 int postIndex = size - 2;
 int start = 0;
 int end = size - 1;
 t -> root -> right = recursiveBuild(inArray, ind + 1, end, postArray,
&postIndex);
 t -> root -> left = recursiveBuild(inArray, start, ind - 1,
postArray, &postIndex);
 return t;
}
#endif
```