西北工業大學

EXPERIMENT REPORT OF DATA STRUCTURE

Experiment 6

NAME : ABID ALI

STUDENT ID : 2019380141

DATE : 06/16/2021

E-Mail : abiduu354@gmail.com

Problem Description:

In this assignment, we are going to write a C/C++ program files(dse_assign06.cpp"and text file with code also included), which that first reads two strings representing the pre-order (Root, Left, Right) and in-order (Left, Root, Right) traversal of a binary tree, and then outputs the post-order (Left, Right, Root) traversal of that tree. We can assume that the input only consists of upper- and lower-case letters and digits.We can also assume that the input is valid. e.g. there exists a binary tree, each of its nodes is labeled by a different character, whose pre-order and in-order are the given strings.This program was completed using Code Blocks.

Goal:

To complete this assessment, you need to complete the implementation of *treeTraversal.c.*
To begin, verify the files needed for this assessment.

1st Step:

Extract the archive to retrieve the files needed to complete this assessment.
Following is an ordered list of steps that serves as a guide to completing this assessment. Work and test incrementally. Save often

2nd Step:

2)First, finish the implementation of function build_tree. In pre-order traversal, we first print the root node. Then traverse it's left sub tree if present. And finally, traverse it's right sub tree if present. In in-order traversal, we traverse first the left sub tree of the node if present. Then print the root node. And finally, traverse it's right sub tree if present. Hence if we have a pre-order traversal, then we can always say that the 0th index element will represent root node of the tree. And if we have a in-order traversal then for every ith index, all the element in the left of it will be present in it's left sub tree and all the elements in the right of it will be in it's right sub tree. Based on these

observations, the algorithm of constructing a binary tree from given in-order and pre-order traversals will be:

① Pick the next element in pre-order traversal (start picking with index 0).
② Create a new node with the data as the picked element.
③ Find the picked element's index from in-order traversal.
④ Recursively call the same function for elements in the left of the picked element and assign it to the left of the picked node.
⑤ Recursively call the same function for elements in the right of the picked element and assign it to the right of the picked node.
⑥ return root node.

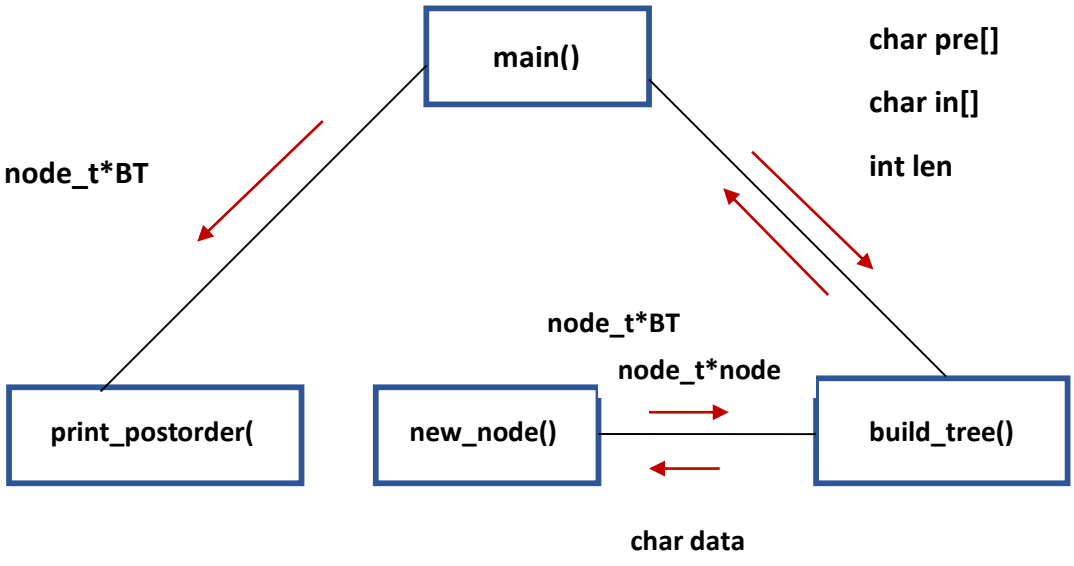
3rd Step:

Secondly, finish the implementation of function print_postorder. This recursive function takes the pointer to the root node as input. If the current root node is not NULL, this function processes first its left sub tree and then right sub tree, finally visit the current root node.

4th Step:

Finally, test your program.

Structure Chart:



Implementation:

In computer science, **tree traversal** (also known as **tree** search and walking the **tree**) is a form of graph **traversal** and refers to the process of visiting (checking and/or updating) each node in a **tree data structure**, exactly once. Such traversals are classified by the order in which the nodes are visited.

1)Linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in many ways to possible. Following is the generally used ways for traversing trees.

```
node_t *new_node(char data)
{
   node_t *node = (node_t *)malloc(sizeof(node_t));
   node->data = data;
   node->left = NULL;
   node->right = NULL;

   return(node);
}
```

2) The struct node that is pointed to the left and right might have other left and right children so we can consider them as sub-trees instead of sub-nodes.

If we look at the structure ,we can see every tree has the following parts:

- A node carrying data
- Two subtrees

3) Uses of Postorder:

Postorder Definition: In a postorder traversal, we recursively do a postorder traversal of the left subtree and the right subtree followed by a visit to the root node.

Our goal is to visit each of the nodes, so we are going to visit all the nodes in the subtree, visiting the root node also visit all the nodes in the right subtree .

```
if (len == 0) {
      return NULL;
   }

   node_t *temp = (node_t *)malloc(sizeof(node_t));
   temp->data = pre[0];
   int k;
```

```
    for (k = 0; k < len; k++) {
        if (in[k] == pre[0]) {
            break;
        }
    }

    temp->left = build_tree(pre+1, in, k);
    temp->right = build_tree(pre+k+1, in+k+1, len-k-1);

    return temp;
}
```

Depending on which order We are going to do this program, there will be always three types of traversals. We are going to do post order for this assignment.The concept post-order were explained by the teacher perfectly to us.I use those those concept in the following problem.

```
void print_postorder(node_t *node)
{

    if (node == NULL)
        return;

    // first recur on left subtree
    print_postorder(node->left);

    // then recur on right subtree
    print_postorder(node->right);

    // now deal with the node
    printf("%c ", node->data);

}
```

Note:

This the algorithm for left-to-right post-order traversal is:

1. Do a post-order traversal of the left subtree
2. Do a post-order traversal of the right subtree
3. Visit the root node (generally output this)

4)

```
{
    char preorder[100], inorder[100];
```

```
    printf("Input the pre-order traversal: ");
    scanf("%s",preorder);
    printf("Input the in-order traversal: ");
    scanf("%s",inorder);

    int len;
    //calculate the length of input strings, the lengths of pre-order and in-order strings are
identical
    for(len=0;;len++)
       if(preorder[len]=='\0') break;

    //build the binary tree
    node_t *BT = build_tree(preorder, inorder, len);

    printf("The post-order traversal is: ");
    //output the post-order of the binary tree
    print_postorder(BT);

    printf("\n");

    return 0;
  }
```

We learned from the book that,A traversal routine is naturally written as a recursive function. Its input parameter is a pointer to a node which we will call rt because each node can be viewed as the root of some subtree. The initial call to the traversal function passes in a pointer to the root node of the tree. The traversal function visits rt and its children (if any) in the desired order. For example, a preorder traversal specifies that rt be visited before its children.

Source of book:
https://books.google.com.hk/books?id=AijEAgAAQBAJ&pg=PA157&lpg=PA157&dq=A+traversal+routine+is+naturally+written+as+a+recursive+function.+Its+input+parameter+is+a+pointer+to+a+node+which+we+will+call+rt+because+each+node+can+be+viewed+as+the+root+of+some+subtree.+The+initial+call+to+the+traversal+function+passes+in+a+pointer+to+the+root+node+of+the+tree.+The+traversal+function+visits+rt+and+its+children+(if+any)+in+the+desired+order.+For+example,+a+preorder+traversal+specifies+that+rt+be+visited+before+its+children&source=bl&ots=0pmgx6gnl5&sig=ACfU3U22xTLxwuRaMgxVx4U18fi2uVJo9A&hl=en&sa=X&ved=2ahUKEwi39cXusJfxAhUZH3AKHZY-Cv8Q6AEwAHoECAMQAw#v=onepage&q=A%20traversal%20routine%20is%20naturally%20written%20as%20a%20recursive%20function.%20Its%20input%20parameter%20is%20a%20pointer%20to%20a%20node%20which%20we%20will%20call%20rt%20because%20each%20node%20can%20be%20viewed%20as%20the%20root%20of%20some%20subtree.%20The%20initial%20call%20to%20the%20traversal%20function%20passes%20in%20a%20pointer%20to%20the%20root%20node%20of%20the%20tree.%20The%20traversal%20function%20visits%20rt%

20and%20its%20children%20(if%20any)%20in%20the%20desired%20order.%20For%20example%2C%20a%20preorder%20traversal%20specifies%20that%20rt%20be%20visited%20before%20its%20children&f=false

5)We begin from the left subtree then we need to visit root node and along with that the right subtree when traversing the tree is complete.

6) We had gone through all the elements, then finally we got our desired post order traversal

Code:

```
/*
 * Binary Tree Traversal.
 *ABID ALI
 * abiduu354@gmail.com
 */
#include <stdio.h>
#include <stdlib.h>
#include "treeTraversal.h"

/** Helper function that allocates a new node with the
  * given data and NULL left and right pointers.
  * We have implemented this function for your convenience.
  * Don't forget to free all memory allocated here before your program terminates. */
node_t *new_node(char data)
{
    node_t *node = (node_t *)malloc(sizeof(node_t));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/** Recursive function to construct a binary tree from preorder traversal pre and inorder traversal in,
```

```
    where the lengths of pre and in are both len. */
node_t *build_tree(char pre[], char in[], int len)
{
    //TODO: Insert your code here
    if (len == 0) {
        return NULL;
    }

    node_t *temp = (node_t *)malloc(sizeof(node_t));//need to allocate memory space for the new node of the tree
    temp->data = pre[0];
    int k;

    for (k = 0; k < len; k++) {
        if (in[k] == pre[0]) {
            break;
        }
    }

    temp->left = build_tree(pre+1, in, k);
    temp->right = build_tree(pre+k+1, in+k+1, len-k-1);

    return temp;

}

/** Prints the post-order traversal of the given root of a binary tree. */
void print_postorder(node_t *node)
{
    //TODO: Insert your code here

    if (node != NULL) {
        print_postorder(node->left);
```

```
        print_postorder(node->right);
        printf("%c", node->data);
    }
}
```

Test Description and Results:

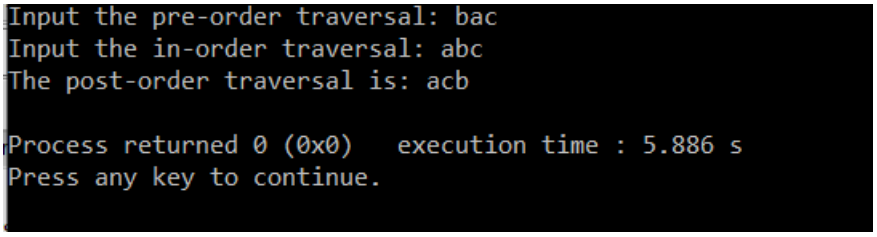
Test 1

Sample input:

Input the pre-order traversal:bac
Input the in-order traversal : abc

Sample output:

The post-order traversal is : acb



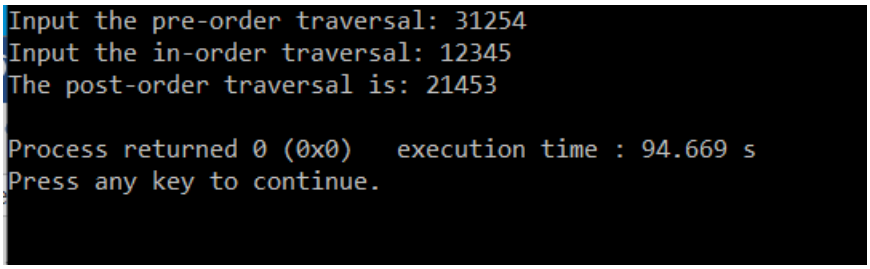
Test 2

Sample input:

Input the pre-order traversal: 31254
Input the in-order traversal : 12345

Sample output:

The post-order traversal is   : 21453



Test 3

Sample input:

Input the pre-order traversal: ABCDEFG
Input the in-order traversal : CBDAFEG

Sample output:

The post-order traversal is : CDBFGEA

```
Input the pre-order traversal: ABCDEFG
Input the in-order traversal: CBDAFEG
The post-order traversal is: CDBFGEA

Process returned 0 (0x0)   execution time : 56.330 s
Press any key to continue.
```

Epilogue:

While doing this assignment I had many bugs which had influence on program working and many troubles with compiling the program, because of syntax and understanding pre-order, in-order traversal and post-order traversal .Fixing those bugs was a challenge for me.I followed teacher practical lecture note,use those idea to solve the problem.Internet was a big help for this program.This program helped me understand the concept of Binary Tree Traversal.

Attachments:

1) treeTraversal.c

treeTraversal.c

2) treeTraversal.txt

3) Assignment 6_ABID ALI_2019380141.pdf

Acknowledgements:

I complete this assignment by myself by using online videos and different books discussing about Algorithm and Data structure. Also,used my knowledge about Object Oriented Programming to solve this problem.It was very useful and helpful for me to increase knowledge for solving complex problem.

Remarks and Grade (by the instructor)

Instructor Signature:

Grading Date: