Assignment 4

Course: Data Structures
Course id:14461002
Student id: 202033762

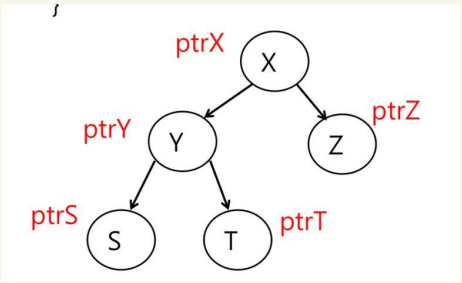
Name: 장민호

Major : 설비소방공학과

Submission Date : 2022_03_24

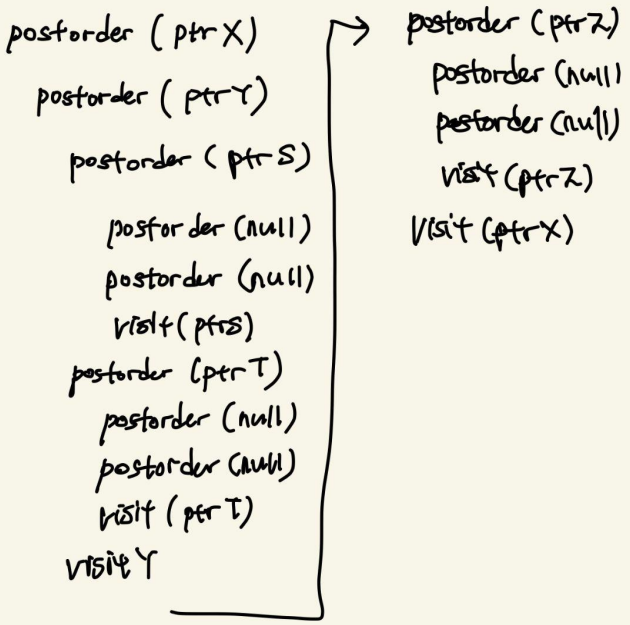
HW 4-1.

HW. 4-1. Recursion Execution

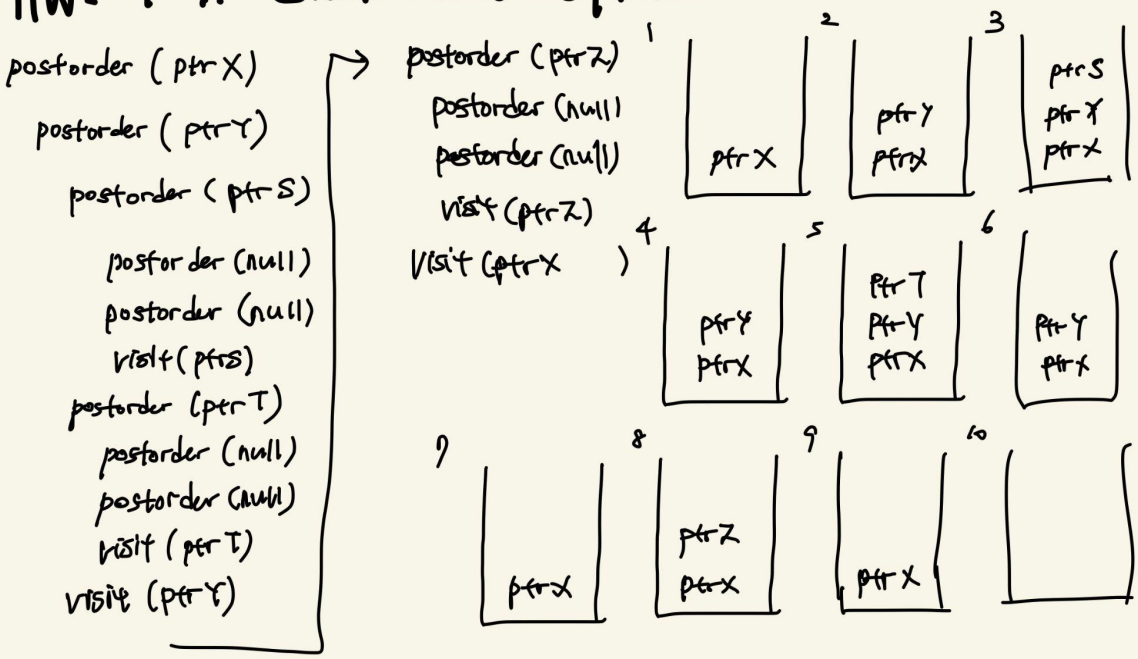



```
void postorder (tree_ptr  ptr)
{ if  (ptr)  {
        postorder (ptr->left_child);
        postorder (ptr->right_child);
        (visit node);}
}
```

ptrX



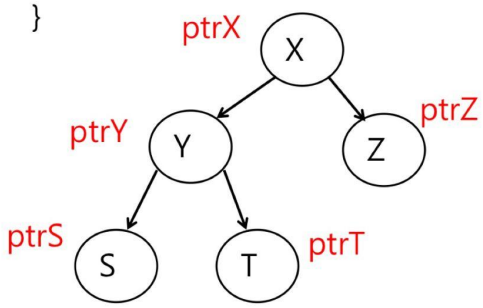
HW. 4-1. Stack State Sequence



HW 4-2.

HW.4-2. Recursion Execution

```
void preorder (tree_ptr  ptr)
{ if  (ptr)  {
        (visit node);
        preorder (ptr->left_child);
        preorder (ptr->right_child);
        }
}
```



preorder (ptrX)
visit (ptrX)
preorder (ptrY)
visit (ptrY)
preorder (ptrS)
visit (ptrS)
preorder (null)
preorder (null)
preorder (ptrT)
visit (ptrT)
preorder (null)
preorder (null)

preorder (ptrZ)
visit (ptrZ)
preorder (null)
preorder (null)

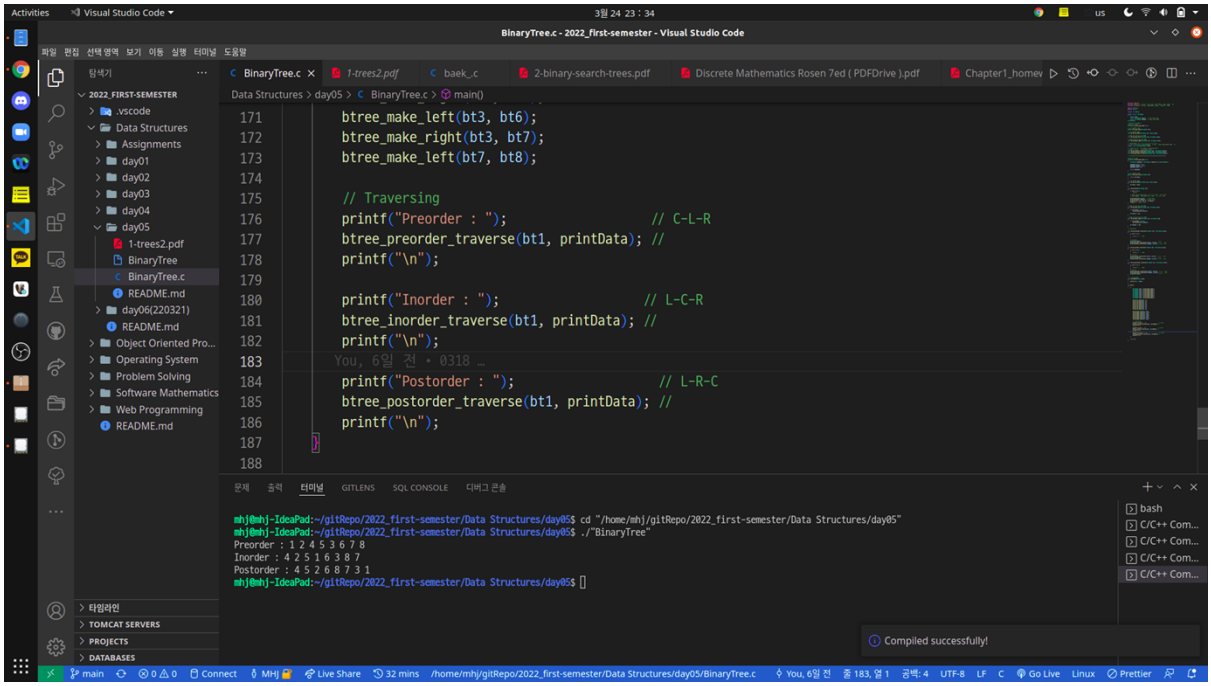
HW 4-3, 4-4.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SUCCESS 1
#define FAIL 0
typedef int BTData;
typedef struct _bTreeNode
{
   BTData data;
   struct _bTreeNode *pLeft;  // left child node
   struct _bTreeNode *pRight; // right child node
} BTreeNode;
/* Binary Tree 동작 */
//트리 노드 생성
BTreeNode *btree_make_node(void);
// 노드 데이터 읽기
BTData btree_get_data(BTreeNode *bt);
// 노드 데이터 쓰기
void btree_set_data(BTreeNode *bt, BTData data);
// bt의 left 에 sub tree 연결
void btree_make_left(BTreeNode *bt, BTreeNode *sub);
```

```
// bt의 right 에 sub tree 연결
void btree_make_right(BTreeNode *bt, BTreeNode *sub);
// Traversing----------------------------------
// 노드를 visit 하여 노드의 데이터에 대해 수행할 함수 (함수포인터) 타입 선언
typedef void fnVistNode(BTData data);
// 주어진 노드 bt 부터 시작하여 traversing 하면서
// node 를 visit 할때마다 action() 수행
void btree_preorder_traverse(BTreeNode *bt, fnVistNode action);
void btree_inorder_traverse(BTreeNode *bt, fnVistNode action);
void btree_postorder_traverse(BTreeNode *bt, fnVistNode action);
// ============================================================
//트리 노드 생성
BTreeNode *btree_make_node(void)
{
   BTreeNode *pNewNode = (BTreeNode *)malloc(sizeof(BTreeNode));
   pNewNode->pLeft = NULL;
   pNewNode->pRight = NULL;
   pNewNode->data = 0;
   return pNewNode;
}
// 노드 데이터 읽기
BTData btree_get_data(BTreeNode *bt)
{
   return bt->data;
}
// 노드 데이터 쓰기
void btree_set_data(BTreeNode *bt, BTData data)
{
   bt->data = data;
}
void btree_delete(BTreeNode *bt)
{
   if (bt == NULL)
       return;
   // 삭제는 post order 방식(L -> R -> C) 순서로 지워야 한다
   // 자기 자신을 지우기 전에 left, right 부터 지워야 한다
   btree_delete(bt->pLeft);  // Left Clear
   btree_delete(bt->pRight); // Right Clear
   printf("Node Clear: %d \n", bt->data);
   free(bt); // Center Clear
}
// bt의 left 에 sub tree 연결
void btree_make_left(BTreeNode *bt, BTreeNode *sub)
{
   if (bt->pLeft != NULL)
       btree_delete(bt->pLeft);
   bt->pLeft = sub;
}
// bt의 right 에 sub tree 연결
void btree_make_right(BTreeNode *bt, BTreeNode *sub)
{
   if (bt->pRight != NULL)
       btree_delete(bt->pRight);
   bt->pRight = sub;
}
// Traversing
void btree_preorder_traverse(BTreeNode *bt, fnVistNode action)
{
```

```
    if (bt == NULL)
    {
        return; // 재귀종료
    }
    // C->L->R
    action(bt->data);                               // C
    btree_preorder_traverse(bt->pLeft, action);  // L:재귀호출
    btree_preorder_traverse(bt->pRight, action); // R:재귀호출
}
void btree_inorder_traverse(BTreeNode *bt, fnVistNode action)
{
    if (bt == NULL)
    {
        return; // 재귀종료
    }
    // L->C->R
    btree_inorder_traverse(bt->pLeft, action);  // L:재귀호출
    action(bt->data);                               // C
    btree_inorder_traverse(bt->pRight, action); // R:재귀호출
}
void btree_postorder_traverse(BTreeNode *bt, fnVistNode action)
{
    if (bt == NULL)
    {
        return; // 재귀종료
    }
    // L>R->C
    btree_postorder_traverse(bt->pLeft, action);  // L:재귀호출
    btree_postorder_traverse(bt->pRight, action); // R:재귀호출
    action(bt->data);                               // C
}
// == fnVisitNode
void printData(BTData data)
{
    printf("%d ", data);
}
int main()
{
    {
        BTreeNode *bt1 = btree_make_node();
        BTreeNode *bt2 = btree_make_node();
        BTreeNode *bt3 = btree_make_node();
        BTreeNode *bt4 = btree_make_node();
        BTreeNode *bt5 = btree_make_node();
        BTreeNode *bt6 = btree_make_node();
        BTreeNode *bt7 = btree_make_node();
        BTreeNode *bt8 = btree_make_node();
        btree_set_data(bt1, 1);
        btree_set_data(bt2, 2);
        btree_set_data(bt3, 3);
        btree_set_data(bt4, 4);
        btree_set_data(bt5, 5);
        btree_set_data(bt6, 6);
        btree_set_data(bt7, 7);
        btree_set_data(bt8, 8);
        btree_make_left(bt1, bt2);
        btree_make_right(bt1, bt3);
        btree_make_left(bt2, bt4);
```

```
        btree_make_right(bt2, bt5);
        btree_make_left(bt3, bt6);
        btree_make_right(bt3, bt7);
        btree_make_left(bt7, bt8);
        // Traversing
        printf("Preorder : ");                    // C-L-R
        btree_preorder_traverse(bt1, printData); //
        printf("\n");
        printf("Inorder : ");                     // L-C-R
        btree_inorder_traverse(bt1, printData); //
        printf("\n");
        printf("Postorder : ");                    // L-R-C
        btree_postorder_traverse(bt1, printData); //
        printf("\n");
    }
    return 0;
}
```



Preorder : 1 2 4 5 3 6 7 8

Inorder : 4 2 5 1 6 3 8 7

Postorder : 4 5 2 6 8 7 3 1