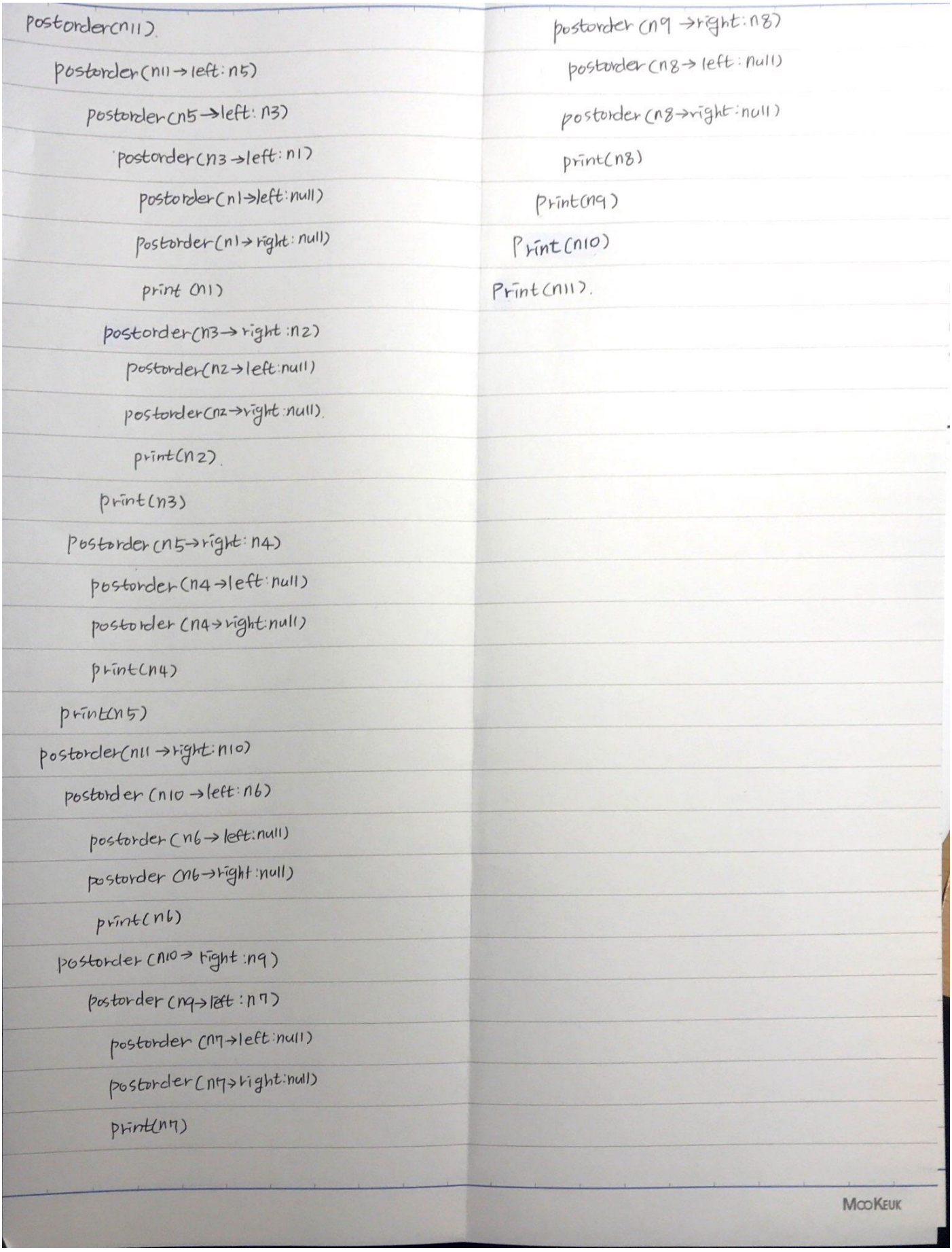
REPORT

Data Structures

Student ID : 1771008
Name : Minjeong Kim

Homework5_1

postorder()의 진행방식은 아래와 같다.



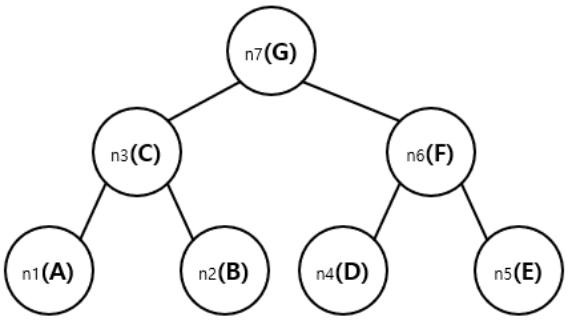
위 종이에 적힌 내용을, 단락 구분과, 함수호출 순서를 보기 쉽게 하기위해 아래와 같이 타이핑 하였다.

```
postorder(n11)
    postorder(n11->left : n5)
        postorder(n5->left : n3)
            postorder(n3->left : n1)
                postorder(n1->left : null)
                postorder(n1->right : null)
                print(n1)
            postorder(n3->right : n2)
                postorder(n2->left : null)
                postorder(n2->right : null)
                print(n2)
            print(n3)
        postorder(n5->right : n4)
            postorder(n4->left : null)
            postorder(n4->right : null)
            print(n4)
        print(n5)
    postorder(n11->right : n10)
        postorder(n10->left : n6)
            postorder(n6->left : null)
            postorder(n6->right : null)
            print(n6)
        postorder(n10->right : n9)
            postorder(n9->left : n7)
                postorder(n7->left : null)
                postorder(n7->right : null)
                print(n7)
            postorder(n9->right : n8)
                postorder(n8->left : null)
                postorder(n8->right : null)
                print(n8)
            print(n9)
        print(n10)
    print(n11)
```

Homework5_2

변수(variable) 분석

TreeNode	
Type	**name**
char	data
TreeNode*	left
TreeNode*	right
TreeNode*	parent



트리 노드 멤버선언
전역변수로 TreeNode를 곧바로 선언 할 시, C의 순차적진행이라는 특징 때문에
노드의 주소값에 대한 메모리 할당이 일어나지 않았다는 오류가 뜬다.
따라서 트리노드의 변수를 전역변수로 먼저 선언해서 주소를 할당하고,
이후 main함수에서 할당된 주소에 각각 트리 노드의 값을 초기화 하도록 코드를 수정하였다.

함수(function) 분석

TreeNode *tree_successor(TreeNode *p)

parameter로 들어오는 TreeNode의 successor값을 찾아서 해당 successor node를 return하는 함수이다.

먼저 tree successor를 찾을 때 두가지 경우로 분리한다.

1. right child node가 있는 경우
   successor를 찾기 위해 parent를 고려할 필요없이 child node를 찾으면 된다.
   inorder traversal 방식이기 때문에, L V R 의 순서로 트리를 탐색해야하므로, right child node가 있는 경우라면 현재 상태가 V의 상태라는 소리다 따라서 R의 상태로 이동하고, 해당 R node의 L 상태에서 탐색이 시작되기 때문에 input node의 right 노드의 left most node로 이동한다.

2. right child node가 없는 경우
   successor를 찾기 위해 parent를 고려해야한다.
   L V R 에서 subtree의 R 탐색이 끝난 경우이기 때문에 해당 node의 parent node로 가서 L V R중 R 단계면 한단계 위 parent node로 가고, 아닐 경우엔 해당 parent node가 successor가 된다.

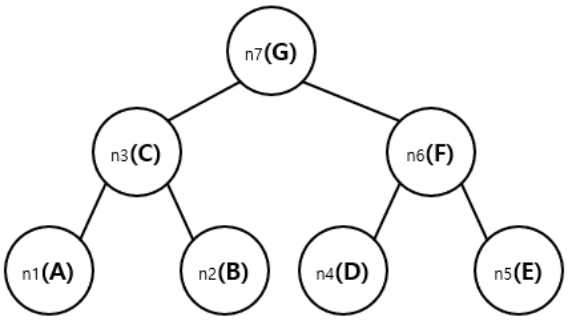
전체 코드(full code)

```c
#include <stdio.h>

typedef struct TreeNode {
    char data;
    struct TreeNode *left, *right, *parent;
} TreeNode;

// G
// C F
// A B D E

TreeNode n1, n2, n3, n4, n5, n6, n7;

TreeNode *tree_successor(TreeNode *p)
{
    TreeNode *x, *y;

    x = p; y = NULL;

    if (x->right != NULL) {// 자식요소가 있다.
        x = x->right;
        while (x->left != NULL) {
            x = x->left;
        }  // x right 노드의 left most node로 이동
        return x;
    }

    y = x->parent;
    while ((y != NULL) && (x == (y->right))) {
        x = y;
        y = y->parent;
    }
    return y;
}

int main()
{
    n1 = { 'A', NULL, NULL, &n3 };
    n2 = { 'B', NULL, NULL, &n3 };
    n3 = { 'C', &n1, &n2, &n7 };
    n4 = { 'D', NULL, NULL, &n6 };
    n5 = { 'E', NULL, NULL, &n6 };
    n6 = { 'F', &n4, &n5, &n7 };
    n7 = { 'G', &n3, &n6, NULL };
    TreeNode *exp = &n7;

    TreeNode *q = exp;

    while (q->left) q = q->left; // Go to the leftmost node

    do
    {
        printf("%c\n", q->data); // Output data
        q = tree_successor(q); // Call the successor
    } while (q); // If not null
}
```

Homework5_3

변수(variable) 분석

TreeNode	
Type	**name**
char	data
TreeNode*	left
TreeNode*	right
TreeNode*	parent



Homework5_2에서 코드를 수정하였다.
트리 노드 멤버선언
전역변수로 TreeNode를 곧바로 선언 할 시, C의 순차적진행이라는 특징 때문에
노드의 주소값에 대한 메모리 할당이 일어나지 않았다는 오류가 뜬다.
따라서 트리노드의 변수를 전역변수로 먼저 선언해서 주소를 할당하고,
이후 main함수에서 할당된 주소에 각각 트리 노드의 값을 초기화 하도록 코드를 수정하였다.

함수(function) 분석

TreeNode *tree_predecessor(TreeNode *p)

parameter로 들어오는 TreeNode의 predecessor값을 찾아서 해당 successor node를 return하는 함수이다.
predecessor을 찾는 방법은 successor 찾는 방법의 반대이기 때문에, find_successor 함수에서 right는 left로 left는 right로 수정하면 find_predecessor 함수가 나온다.
inorder traversal 에서 find_successor의 방식이, L V R이었고, find_predecessor의 경우에는 R V L 이기 때문이다.

그래도 tree predecessor를 찾는 두가지 경우로 분리하자면

1. left child node가 있는 경우
   predecessor를 찾기 위해 parent를 고려할 필요없이 child node를 찾으면 된다.
   inorder traversal 방식의 반대이기 때문에, R C L 의 순서로 트리를 탐색해야하므로, left child node가 있는 경우라면 현재 상태가 V의 상태라는 소리다 따라서 L의 상태로 이동하고, 해당 L node의 R 상태에서 탐색이 시작되기 때문에 input node의 left 노드의 right most node로 이동한다.

2. right child node가 없는 경우
   predecessor를 찾기 위해 parent를 고려해야한다.
   R V L 에서 subtree의 L 탐색이 끝난 경우이기 때문에 해당 node의 parent node로 가서 R V L중 L 단계면 한단계 위 parent node로 가고, 아닐 경우엔 해당 parent node가 successor가 된다.

전체 코드(full code)

```c
#include <stdio.h>


typedef struct TreeNode {
    char data;
    struct TreeNode *left, *right, *parent;
} TreeNode;

// G
// C F
// A B D E

TreeNode n1, n2, n3, n4, n5, n6, n7;

TreeNode *tree_predecessor(TreeNode *p)
{
    TreeNode *x, *y;

    x = p; y = NULL;

    if (x->left != NULL) {// 자식요소가 있다.
        x = x->left;
        while (x->right != NULL) {
            x = x->right;
        }  // x left 노드의 right most node로 이동
        return x;
    }

    y = x->parent;
    while ((y != NULL) && (x == (y->left))) {
        x = y;
        y = y->parent;
    }
    return y;
}

int main()
{
    n1 = { 'A', NULL, NULL, &n3 };
    n2 = { 'B', NULL, NULL, &n3 };
    n3 = { 'C', &n1, &n2, &n7 };
    n4 = { 'D', NULL, NULL, &n6 };
    n5 = { 'E', NULL, NULL, &n6 };
    n6 = { 'F', &n4, &n5, &n7 };
    n7 = { 'G', &n3, &n6, NULL };
    TreeNode *exp = &n7;

    TreeNode *q = exp;

    while (q->right) q = q->right; // Go to the leftmost node

    do
    {
        printf("%c\n", q->data); // Output data
        q = tree_predecessor(q); // Call the successor
    } while (q); // If not null
}
```