

자료구조 보고서

Homework#8

학과 : 소프트웨어학과

학번 : 2016039040

이름 : 윤용진

1. Binary Search Tree의 기능 구현에 관한 것이다.

(a) binary-search-tree-1.c의 다음 함수를 완성한다.

```
void inorderTraversal(Node* ptr); /* recursive inorder traversal */
void preorderTraversal(Node* ptr); /* recursive preorder traversal */
void postorderTraversal(Node* ptr); /* recursive postorder traversal */
int insert(Node* head, int key); /* insert a node to the tree */
int deleteLeafNode(Node* head, int key); /* delete the leaf node for the key
*/
Node* searchRecursive(Node* ptr, int key); /* search the node for the key
*/
Node* searchIterative(Node* head, int key); /* search the node for the key
*/
int freeBST(Node* head); /* free all memories allocated to the tree */
```

(b) 이해한 부분을 주석으로 남긴다.

```
/*
 * Binary Search Tree #1
 *
 * Data Structures
 *
 * Department of Computer Science
 * at Chungbuk National University
 *
 */

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

typedef struct node {
    int key;
    struct node *left;
    struct node *right;
} Node;

int initializeBST(Node** h);

/* functions that you have to implement */
void inorderTraversal(Node* ptr); /* recursive inorder traversal */
void preorderTraversal(Node* ptr); /* recursive preorder traversal */
void postorderTraversal(Node* ptr); /* recursive postorder traversal */
int insert(Node* head, int key); /* insert a node to the tree */
```

```

int deleteLeafNode(Node* head, int key); /* delete the leaf node for the key */
Node* searchRecursive(Node* ptr, int key); /* search the node for the key */
Node* searchIterative(Node* head, int key); /* search the node for the key */
int freeBST(Node* head); /* free all memories allocated to the tree */

/* you may add your own defined functions if necessary */

int precheck(Node *head,int);

int recursiveDeletechecker = 0;
Node* node2search = NULL;

#define SEARCHMODE 1

int main()
{
    char command;
    int key;
    Node* head = NULL;
    Node* ptr = NULL; /* temp */

    printf("----- [Yoon YongJin] [2016039040] -----\\n");
    do{
        printf("\\n\\n");

        printf("-----\\n");
        printf("                                Binary Search Tree #1
\\n");

        printf("-----\\n");
        printf("      Initialize      BST                                = z
\\n");
        printf(" Insert Node                = n      Delete Leaf Node
= d \\n");
        printf(" Inorder Traversal    = i      Search Node Recursively
= s \\n");
        printf(" Preorder Traversal   = p      Search Node Iteratively
= f\\n");
        printf(" Postorder Traversal   = t      Quit
= q\\n");

```

```

printf("-----\n");

printf("Command = ");
scanf(" %c", &command);

switch(command) {
case 'z': case 'Z':
    initializeBST(&head);
    break;
case 'q': case 'Q':
    freeBST(head);
    break;
case 'n': case 'N':
    printf("Your Key = ");
    scanf("%d", &key);
    insert(head, key);
    break;
case 'd': case 'D':
    printf("Your Key = ");
    scanf("%d", &key);
    deleteLeafNode(head, key);
    break;
case 'f': case 'F':
    printf("Your Key = ");
    scanf("%d", &key);
    ptr = searchIterative(head, key);
    if(ptr != NULL)
        printf("\n node [%d] found at %p\n", ptr->key,
ptr);
    else
        printf("\n Cannot find the node [%d]\n", key);
    break;
case 's': case 'S':
    printf("Your Key = ");
    scanf("%d", &key);
    if (!precheck(head, SEARCHMODE)) break;
    node2search = NULL;
    ptr = searchRecursive(head->left, key);
    if(ptr != NULL)

```

```

printf("\n node [%d] found at %p\n", ptr->key,
ptr);

else
printf("\n Cannot find the node [%d]\n", key);
break;

case 'i': case 'I':
if (precheck(head, SEARCHMODE))
{
/*
* Traversal 함수들은 head가 아닌 head->left를 인
자로 받기 때문에 초기화 되지 않은 경우,
* NULL포인터 호출로 에러가 발생할 수 있어, 함수
호출 전에 전처리
*/
inorderTraversal(head->left);
}
break;
case 'p': case 'P':
if (precheck(head, SEARCHMODE))
{
preorderTraversal(head->left);
}
break;
case 't': case 'T':
if (precheck(head, SEARCHMODE))
{
postorderTraversal(head->left);
}
break;
default:
printf("\n      >>>>   Concentration!!   <<<<<   \n");
break;
}

}while(command != 'q' && command != 'Q');

return 1;
}

```

```

/* 전처리 함수 */
int precheck(Node* head, int mode)
{
    /* 헤드노드 초기화 */
    if (head == NULL)
    {
        printf("Initialize First\n");
        return 0;
    }
    switch (mode)
    {
        /* 트리를 탐색해야하는 경우의 전처리 - 트리에 생성된 노드가 없는 경
우 */
        case SEARCHMODE:
            if (head->right == head && head->left == NULL)
            {
                printf("There is no Tree\n");
                return 0;
            }
            break;
        default:
            break;
    }
    return 1;
}

/* 초기화 */
int initializeBST(Node** h) {

    /* if the tree is not empty, then remove all allocated nodes from the
tree*/
    if(*h != NULL)
        freeBST(*h);

    /* create a head node */
    *h = (Node*)malloc(sizeof(Node));
    (*h)->left = NULL;      /* root */
    (*h)->right = *h;
    (*h)->key = -9999;
    return 1;
}

```

```

}

/* 중위 순회 */
void inorderTraversal(Node* ptr)
{
    if (ptr)
    {
        /* 왼쪽노드를 먼저 탐색 */
        inorderTraversal(ptr->left);
        /* 오른쪽 노드를 탐색하기 전에 현재 노드를 표기 */
        printf("[%d] ", ptr->key);
        /* 오른쪽 노드 탐색 */
        inorderTraversal(ptr->right);
    }
}

/* 전위 순회 */
void preorderTraversal(Node* ptr)
{
    if (ptr)
    {
        /* 현재 노드를 표기 */
        printf("[%d] ", ptr->key);
        /* 왼쪽 노드 탐색 */
        preorderTraversal(ptr->left);
        /* 오른쪽 노드 탐색 */
        preorderTraversal(ptr->right);
    }
}

/* 후위 순회 */
void postorderTraversal(Node* ptr)
{
    if (ptr)
    {
        /* 왼쪽 노드 탐색 */
        postorderTraversal(ptr->left);
        /* 오른쪽 노드 탐색 */
        postorderTraversal(ptr->right);
        /* 현재 노드를 표기 */
    }
}

```

```

        printf("[%d] ", ptr->key);
    }
}

/* 노드 삽입 */
int insert(Node* head, int key)
{
    /* 전처리 */
    if (!precheck(head, 0)) return 0;

    Node *p;

    /* 새로운 노드 생성 */
    Node* node2insert = (Node*)malloc(sizeof(Node));
    node2insert->key = key;
    node2insert->left = NULL;
    node2insert->right = NULL;

    p = head;

    while (1)
    {
        /* 현재노드가 헤드노드인 경우 */
        if (p->key == -9999)
        {
            /* 트리에 헤드노드를 제외한 노드가 없는 경우 새로운 노드를
헤드노드 왼쪽에 삽입 */
            if (p->left == NULL)
            {
                p->left = node2insert;
                return 0;
            }
            /* 트리에 기존 노드가 있는 경우 다음 노드 탐색 */
            p = p->left;
        }

        /* 입력받은 키가 현재 노드의 키 값보다 작다면 */
        if (p->key > key)
        {
            if (p->left == NULL)

```



```

        {
            /* 비어있다면 노드 삽입 */
            p->left = node2insert;
            return 0;
        }
        /* 좌측에 기존 노드가 있는 경우 다음 노드 탐색 */
        p = p->left;
    }
    /* 입력받은 키가 현재 노드의 키 값보다 크다면 */
    else if (p->key < key)
    {
        if (p->right == NULL)
        {
            /* 비어있다면 노드 삽입 */
            p->right = node2insert;
            return 0;
        }
        /* 우측에 기존 노드가 있는 경우 다음 노드 탐색 */
        p = p->right;
    }
    /* 입력받은 키가 현재 노드의 키 값과 같다면 */
    else if (p->key == key)
    {
        /* 왼쪽노드를 우선 확인하고 비어 있으면 삽입 */
        if (p->left == NULL)
        {
            p->left = node2insert;
            return 0;
        }
        /* 오른쪽 노드를 확인하고 비어 있으면 새로운 노드 삽입 */
        else if (p->right == NULL)
        {
            p->right = node2insert;
            return 0;
        }
        /* 양쪽에 기존 노드가 있다면 탐색을 이어 나간다. */
        p = p->left;
    }
}

```

```

        return 0;
    }

    /* 잎노드 삭제 */
    int deleteLeafNode(Node* head, int key)
    {
        /* 전처리, 헤드가 초기화 되지 않은 경우 리턴 */
        if (!precheck(head, 0)) return 0;

        /* 현재노드가 헤드노드면 왼쪽만 탐색 */
        if (head->right == head)
        {
            /*
             * 현재노드가 헤드노드면 현재 함수는 모든 노드를 탐색한 후 리턴되므
로,
             * 다음 노드를 탐색하기 전에 전역변수 recursiveDeletechecker를 0으
로 초기화.
            */
            recursiveDeletechecker = 0;

            /*
             * 탐색중인 노드가 삭제되었을 경우에는 0(NULL)값을 리턴을 통해 저장
하여 줄기 노드를 잎 노드화,
             * 삭제되지 않은 경우에는 해당 노드를 리턴을 통해 저장하여 기존의 연
결 유지
            */
            /* 함수의 반환 자료형이 Node*여야 한다. */
            /*
            //head->left = deleteLeafNode(head->left, key);

            if (deleteLeafNode(head->left, key))
            {
                head->left = NULL;
            }
            */
            /* 양쪽노드가 NULL값이 아니면 잎노드가 아니므로 계속 탐색 */
            else
            {
                if (head->left != NULL)
                {
                    //head->left = deleteLeafNode(head->left, key);

```

```

        if (deleteLeafNode(head->left, key))
        {
            head->left = NULL;
        }
    }
    if (head->right != NULL)
    {
        //head->right = deleteLeafNode(head->right, key);
        if (deleteLeafNode(head->right, key))
        {
            head->right = NULL;
        }
    }
}
/* 현재 노드의 키 값이 입력받은 키 값과 같고, 아직 삭제된 노드가 없다면 */
if (head->key == key && recursiveDeletechecker == 0)
{
    /* 재귀함수의 특성상 상태를 기억하지 못하므로 전역변수
recursiveDeletechecker에 1을 저장해, 노드의 삭제 여부를 기억한다. */
    recursiveDeletechecker = 1;
    /* 현재 노드가 잎 노드라면 */
    if (head->left == NULL && head->right == NULL)
    {
        /* 삭제하고, 0(NULL)을 반환 */
        free(head);
        //return NULL;
        return 1;
    }
    /* 삭제할 노드가 잎 노드가 아닌 경우 */
    else
    {
        printf("the node[%d] is not a leaf\n", key);
        //return head;
        return 0;
    }
}

/* 모든 탐색을 마치고(현재노드가 헤드노드) 나서도, 삭제된 노드가 없다면 */
if(head->key == -9999 && recursiveDeletechecker == 0)
{

```

```

        printf("Cannot find the node[%d]\n", key);
        //return head;
        return 0;
    }
    //return NULL;
    return 0;
}

/* 재귀 탐색 */
Node* searchRecursive(Node* ptr, int key)
{
    /* 전처리 */
    if (!precheck(ptr, SEARCHMODE)) return 0;

    /* 재귀 탐색 */
    if (ptr->left) searchRecursive(ptr->left, key);
    if (ptr->right) searchRecursive(ptr->right, key);

    /* 현재 노드가 찾던 노드이면 */
    if (ptr->key == key)
    {
        /* 전역변수 node2search에 현재노드를 저장 */
        node2search = ptr;
    }
    /* node2search가 NULL이 아니면 리턴*/
    if (node2search) return node2search;
    else return 0;
}

/* 반복 탐색 */
Node* searchIterative(Node* head, int key)
{
    /* 전처리 */
    if (!precheck(head, SEARCHMODE)) return 0;

    /* 반복탐색 */
    while (head)
    {
        /* 현재노드가 헤드노드이면 왼쪽노드 탐색 */
        if (head->right == head) head = head->left;
    }
}

```

```

        /* key값이 일치하는 노드 탐색 성공 시 리턴 */
        if (key == head->key) return head;
        /* 키값이 작으면 왼쪽노드 탐색 */
        if (key < head->key) head = head->left;
        /* 키값이 크면 오른쪽 노드 탐색 */
        else head = head->right;
    }

    return 0;
}

/* 재귀적으로 트리를 탐색하여 메모리 해제*/
int freeBST(Node* head)
{
    /* 전처리, 헤드가 초기화 되지 않은경우 리턴      */
    if (!precheck(head,0)) return 0;

    /* 현재노드가 헤드노드면 왼쪽만 탐색 */
    if (head->right == head)
    {
        freeBST(head->left);
    }
    /* 양쪽노드가 NULL값이 아니면 메모리 해제 */
    else
    {
        if(head->left != NULL) freeBST(head->left);
        if(head->right != NULL) freeBST(head->right);
    }
    free(head);

    return 0;
}

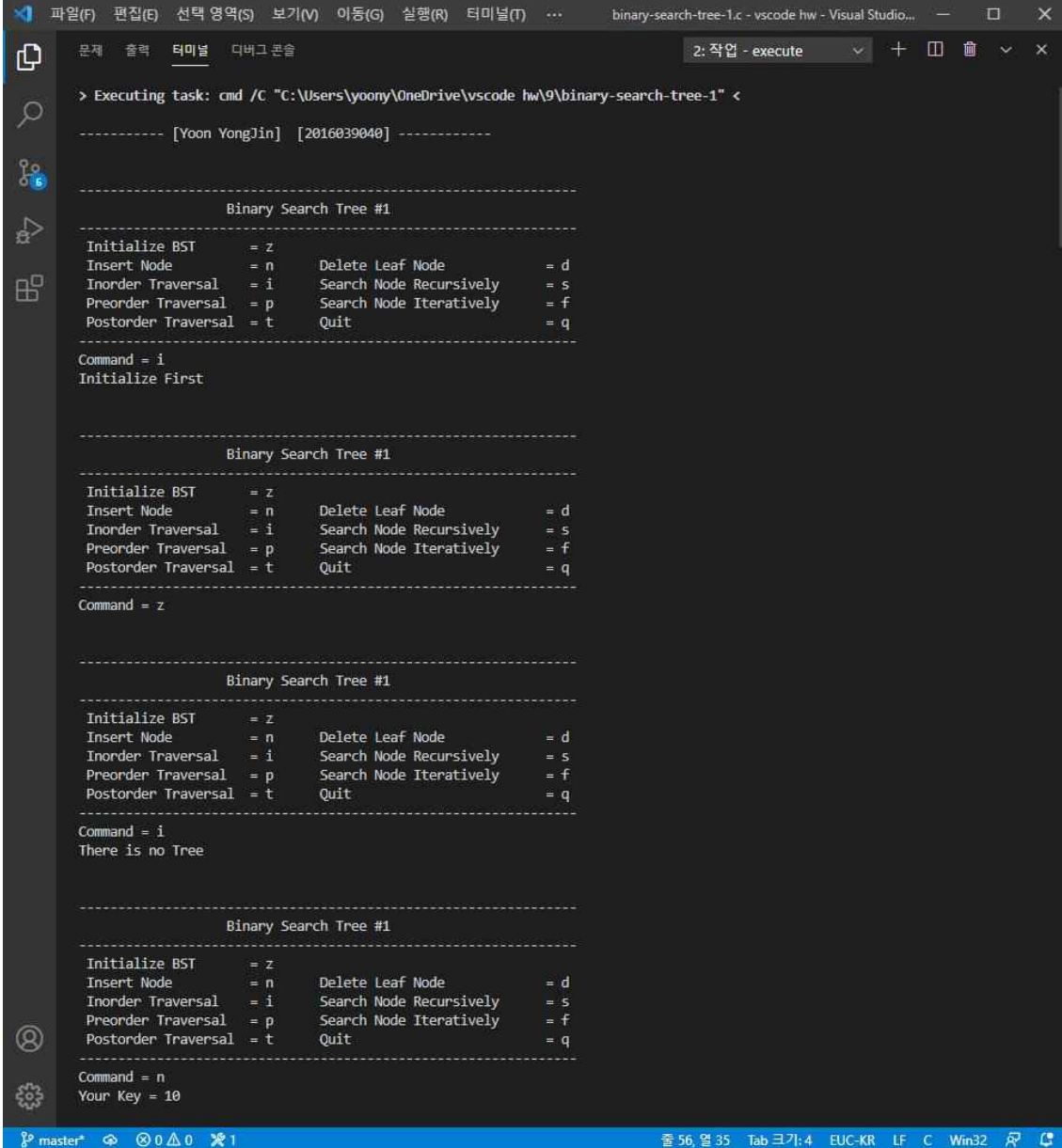
```

4. GitHub에 hw7 Repository를 생성하고 doubly-linked-list.c를 업로드 한다.

<https://github.com/uniz21/DataStructure-HW-9>

7. 보고서에 실행결과를 Screen Capture하여 첨부한다.

실행결과



```
binary-search-tree-1.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 2: 작업 - execute
> Executing task: cmd /C "C:\Users\yoony\OneDrive\vscode hw\9\binary-search-tree-1" <
----- [Yoon YongJin] [2016039040] -----

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit              = q
-----
Command = i
Initialize First

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit              = q
-----
Command = z

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit              = q
-----
Command = i
There is no Tree

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit              = q
-----
Command = n
Your Key = 10
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-1.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 2: 작업 - execute + - ✕
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit              = q
-----
Command = i
[10]
-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit              = q
-----
Command = n
Your Key = 8
-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit              = q
-----
Command = i
[8] [10]
-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit              = q
-----
Command = n
Your Key = 15
-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node      = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit              = q
-----
master* 56, 열 35 Tab 크기: 4 EUC-KR LF C Win32
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-1.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 2: 작업 - execute
Preorder Traversal = p Search Node Iteratively = f
Postorder Traversal = t Quit = q
-----
Command = n
Your Key = 30

-----
Binary Search Tree #1
-----
Initialize BST = z
Insert Node = n Delete Leaf Node = d
Inorder Traversal = i Search Node Recursively = s
Preorder Traversal = p Search Node Iteratively = f
Postorder Traversal = t Quit = q
-----
Command = i
[8] [10] [15] [30]

-----
Binary Search Tree #1
-----
Initialize BST = z
Insert Node = n Delete Leaf Node = d
Inorder Traversal = i Search Node Recursively = s
Preorder Traversal = p Search Node Iteratively = f
Postorder Traversal = t Quit = q
-----
Command = p
[10] [8] [15] [30]

-----
Binary Search Tree #1
-----
Initialize BST = z
Insert Node = n Delete Leaf Node = d
Inorder Traversal = i Search Node Recursively = s
Preorder Traversal = p Search Node Iteratively = f
Postorder Traversal = t Quit = q
-----
Command = t
[8] [30] [15] [10]

-----
Binary Search Tree #1
-----
Initialize BST = z
Insert Node = n Delete Leaf Node = d
Inorder Traversal = i Search Node Recursively = s
Preorder Traversal = p Search Node Iteratively = f
Postorder Traversal = t Quit = q
-----
Command = i
[8] [10] [15] [30]
```



```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-1.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 2: 작업 - execute + □ ✖ ▼ ×

Insert Node      = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = d
Your Key = 10
the node[10] is not a leaf

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = d
Your Key = 15
the node[15] is not a leaf

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = d
Your Key = 30

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = i
[8] [10] [15]

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-1.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 2: 작업 - execute
Insert Node      = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = s
Your Key = 8

node [8] found at 00C91620

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = s
Your Key = 10

node [10] found at 00C91608

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = s
Your Key = 15

node [15] found at 00C91638

-----
Binary Search Tree #1
-----
Initialize BST    = z
Insert Node       = n      Delete Leaf Node    = d
Inorder Traversal = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively = f
Postorder Traversal = t      Quit                = q
-----
Command = f
Your Key = 8

node [8] found at 00C91620
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-1.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 2: 작업 - execute
-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node          = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit                      = q
-----
Command = f
Your Key = 8

node [8] found at 00C91620

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node          = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit                      = q
-----
Command = f
Your Key = 10

node [10] found at 00C91608

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node          = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit                      = q
-----
Command = f
Your Key = 15

node [15] found at 00C91638

-----
Binary Search Tree #1
-----
Initialize BST      = z
Insert Node        = n      Delete Leaf Node          = d
Inorder Traversal  = i      Search Node Recursively = s
Preorder Traversal = p      Search Node Iteratively  = f
Postorder Traversal = t      Quit                      = q
-----
Command =
```