

# 자료구조 보고서

Homework#10

학과 : 소프트웨어학과

학번 : 2016039040

이름 : 윤용진

1. Binary Search Tree의 기능 구현 #2에 관한 것이다.

(a) binary-search-tree-2.c의 다음 함수를 완성한다.

```
void recursiveInorder(Node* ptr); /* recursive inorder traversal */
void iterativeInorder(Node* ptr); /* iterative inorder traversal p.224 */
void levelOrder(Node* ptr); /* level order traversal p.225 */
int deleteNode(Node* head, int key); /* delete the node for the key: 3
cases */
Node* pop(); /* for stack */
void push(Node* aNode);
Node* deQueue(); /* for queue */
void enqueue(Node* aNode);
```

(b) deleteNode(Node\* head, int key) 함수는 총 3가지 사항에 대해 구현되어야 한다.

- i. 삭제하고자 하는 노드가 단말노드 일때
- ii. 삭제하고자 하는 노드가 하나의 자식만을 가질 때
- iii. 삭제하고자 하는 노드가 두개의 자식을 가질 때. 이 경우 오른쪽 서브트리에서 가장 작은 값으로 대체되도록 한다.

(c) 이해한 부분을 주석으로 남긴다.

```
/*
 * Binary Search Tree #2
 *
 * 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;

/* for stack */
#define MAX_STACK_SIZE 20
Node* stack[MAX_STACK_SIZE];
int top = -1;
```

```

Node* pop();
void push(Node* aNode);

/* for queue */
#define MAX_QUEUE_SIZE          20
Node* queue[MAX_QUEUE_SIZE];
int front = -1;
int rear = -1;

Node* deQueue();
void enQueue(Node* aNode);

int initializeBST(Node** h);

/* functions that you have to implement */
void recursiveInorder(Node* ptr);      /* recursive inorder traversal */
void iterativeInorder(Node* ptr);      /* iterative inorder traversal */
void levelOrder(Node* ptr);           /* level order traversal */
int insert(Node* head, int key);       /* insert a node to the tree */
int deleteNode(Node* head, int key);   /* delete 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 */
Node* minSub(Node* ptr);
//void printStack();

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

    printf("[----- [Yoon Yong Jin] [2016039040] -----]");

    do{
        printf("\n\n");

        printf("-----\n");
        printf("
Binary Search Tree #2

```

```

        \n");

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

        printf("        Initialize        BST                =        z
        \n");

        printf("        Insert Node                = i                Delete Node
= d \n");

        printf("        Recursive Inorder        = r                Iterative Inorder (Stack)
= t \n");

        printf("        Level Order (Queue)        = l                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 'i': case 'I':
                printf("Your Key = ");
                scanf("%d", &key);
                insert(head, key);
                break;
        case 'd': case 'D':
                printf("Your Key = ");
                scanf("%d", &key);
                deleteNode(head, key);
                break;

        case 'r': case 'R':
                /* head가 초기화되지 않았다면 head->left를 참조할수 없다
*/

                if (head == NULL)
                {

```

```

        printf("Initialize First");
        break;

    }
    else
    {
        recursiveInorder(head->left);
        break;
    }
case 't': case 'T':
    /* head가 초기화되지 않았다면 head->left를 참조할수 없다
*/
    if (head == NULL)
    {
        printf("Initialize First");
        break;
    }
    else
    {
        iterativeInorder(head->left);
        break;
    }

case 'l': case 'L':
    /* head가 초기화되지 않았다면 head->left를 참조할수 없다
*/
    if (head == NULL)
    {
        printf("Initialize First");
        break;
    }
    else
    {
        levelOrder(head->left);
        break;
    }

default:
    printf("\n      >>>>>   Concentration!!   <<<<<   \n");
    break;
}

```

```

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

        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;

        /* 스택 초기화 */
        top = -1;

        /* 큐 초기화 */
        front = rear = -1;

        return 1;
    }

    void recursiveInorder(Node* ptr)
    {
        if(ptr) { /* 현재 노드가 NULL이 아니면 */
            recursiveInorder(ptr->left); /* 왼쪽 노드 재귀탐색 */
            printf(" [%d] ", ptr->key); /* 출력 */
            recursiveInorder(ptr->right); /* 오른쪽 노드 재귀탐색 */
        }
    }

    /**

```

```

* textbook: p 224s
*/
void iterativeInorder(Node* node)
{
    /* 트리에 노드가 없는 경우 종료 */
    if (node == NULL)
    {
        printf("There is no Tree");
        return;
    }

    /* 무한반복 */
    for (;;) {
        /* 중위표기 */
        for (; node; node = node->left)/* 현재 노드가 NULL이 아니면, */
            push(node); /* 스택에 삽입 하고 왼쪽 노드 탐색한다 */

        node = pop(); /* 스택에서 노드를 꺼내, */
        if (!node) break; /* 공백 스택이면 종료, */
        printf(" [%d] ", node->key);/* 공백 스택이 아니면 출력한다 */

        node = node->right;/* 오른쪽 노드 탐색 */
    }
}

/**
* textbook: p 225
*/
void levelOrder(Node* ptr)
{
    /* 트리에 노드가 없는 경우 종료 */
    if (ptr == NULL)
    {
        printf("There is no Tree");
        return;
    }

    front = -1;
    rear = -1;

```

```

/* 레벨 순서 트리 순회 */
if (!ptr) return; /* 공백 트리 */

enqueue(ptr); /* 큐에 노드 삽입 */

/* 무한반복 */
for ( ; ; ) {
    ptr = dequeue(); /* 큐에서 삭제한 노드를 ptr에 저장 */

    if (ptr) { /* ptr이 공백노드(NULL)이 아니면 */
        printf(" [%d] ", ptr->key); /* 출력 */

        if (ptr->left) /* 왼쪽 노드가 있다면 */
            enqueue(ptr->left); /* 큐에 삽입 */
        if (ptr->right) /* 오른쪽 노드가 있다면 */
            enqueue(ptr->right); /* 큐에 삽입 */
    }
    else break; /* ptr이 공백 노드면 종료 */
}
}

int insert(Node* head, int key)
{
    /* head가 초기화되지 않은경우 */
    if (head == NULL)
    {
        printf("Initialize First");
        return 0;
    }

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

    /* 트리에 노드가 없다면 */
    if (head->left == NULL) {
        /* root노드로 삽입 */

```



```

        head->left = newNode;
        return 1;
    }

    /* head->left is the root */
    Node* ptr = head->left;

    /* 탐색 중 이전 부모 노드를 기억하기 위한 노드 */
    Node* parentNode = NULL;

    while(ptr != NULL) {

        /* if there is a node for the key, then just return */
        if(ptr->key == key) return 1;

        /* we have to move onto children nodes,
         * keep tracking the parent using parentNode */
        parentNode = ptr;

        /* key comparison, if current node's key is greater than input
key
         * then the new node has to be inserted into the right subtree;
         * otherwise the left subtree.
         */
        if(ptr->key < key)
            ptr = ptr->right;
        else
            ptr = ptr->left;
    }

    /* linking the new node to the parent */
    if(parentNode->key > key)
        parentNode->left = newNode;
    else
        parentNode->right = newNode;
    return 1;
}

int deleteNode(Node* head, int key)

```

```

{
    /* head가 초기화되지 않은경우 */
    if (head == NULL)
    {
        printf("Initialize First");
        return 0;
    }
    /* root노드가 NULL인 경우 */
    else if (head->left == NULL)
    {
        printf("There is no Tree");
        return 0;
    }

    Node* p = head;
    Node* parentNode = NULL; /* 삭제한 노드의 부모노드 */
    Node* childNode = NULL; /* 삭제한 노드를 대체할 자식노드 */
    int direction = 0; /* 삭제한 노드의 방향 */

    while (p != NULL)
    {
        /* 삭제해야할 노드가 root노드인 경우, 자식노드가 있다면 root노드를
대체해야 하므로
        * 첫 시작 시 head노드를 부모노드로 설정한다. */
        if (p->right == p)/* 현재 노드가 head라면 */
        {
            parentNode = p; /* head를 부모노드로 설정 */
            direction = -1; /* head노드는 왼쪽에 root 노드를 갖는다
*/

            p = p->left; /* root노드 이어서 탐색 */
            continue;
        }

        if (key == p->key) /* 삭제할 노드 발견시 */
        {
            /* 리프 노드의 삭제 -> 그냥 삭제*/
            if ((p->left || p->right) == NULL)/* 두 자식노드가 모두
NULL이어야 한다. */
            {
                free(p);/* 메모리 해제 */
            }
        }
    }
}

```

```

/* 삭제한 노드가 부모노드의 어느 방향에 있던 자식
노드인지 확인 */
/* 삭제한 노드가 부모노드의 어느 방향에 있던 자식
if (direction == 1)/* 오른쪽 노드 */
    parentNode->right = NULL;/* 공백처리 */
else if (direction == -1)/* 왼쪽 노드*/
    parentNode->left = NULL;/* 공백처리 */

return 1;
}
/* 하나의 자식을 가진 줄기 노드 삭제 -> 자식노드로 대체 */
/* 두 자식노드가 모두 NULL인 경우를 위에서 지나왔으므로
적어도 하나 이상의 자식노드가 NULL이 아니다. */
else if ((p->left && p->right) == NULL)/* AND 논리 연산을
통해 두 노드중 하나만 NULL인 경우 */
{
    /* 자식노드의 위치를 확인 */
    if (p->left != NULL) /* 왼쪽 */
        childNode = p->left;/* 대체할 자식 노드 설
정 */
    else/* 오른쪽 */
        childNode = p->right;/* 대체할 자식 노드
설정 */

    free(p);/* 메모리 해제 */

/* 삭제한 노드가 부모노드의 어느 방향에 있던 자식
노드인지 확인 */
if (direction == 1)/* 오른쪽 노드 */
    parentNode->right = childNode;/* 설정해
둔 자식노드로 대체 */
else if (direction == -1)/* 왼쪽 노드 */
    parentNode->left = childNode;/* 설정해
둔 자식노드로 대체 */

return 1;
}
/* 두개의 자식을 가진 줄기 노드 삭제
* -> 왼쪽 서브트리의 가장 큰 원소 혹은 오른쪽 서브트리의
가장 작은 원소로 root를 대체(과제는 오른쪽)

```

```

        * (이원탐색 트리의 특성을 유지하기 위해) */
        /* 자식노드들이 (NULL,NULL), (left,NULL), (NULL, right)인
경우를 모두 확인한 뒤이므로 모든 자식노드들이 비어있지 않은 상태 */
        else
        {
            /* 오른쪽 서브트리의 가장 작은 키 값을 갖는 노드를
추출 */
            childNode = minSub(p->right); /* 자식노드로 설정
*/

            if (childNode == p->right) /* 대체할 자식노드가 직
계자식(왼쪽노드 NULL인 서브트리)이면 */
            {
                if (childNode->right != NULL) /* 자식노드
의 오른쪽 노드가 있다면 */
                    childNode->right =
p->right->right; /* 연결 */
            }
            /* 대체할 자식노드가 서브트리의 잎 노드이면 */
            else
                childNode->right = p->right; /* 대체를 위
해 설정된 자식노드의 오른쪽에 삭제할 노드의 기존 자식 노드들을 연결 */

            childNode->left = p->left; /* 왼쪽 노드 연결 */

            free(p); /* 메모리 해제 */

            /* 삭제한 노드가 부모노드의 어느 방향에 있던 자식
노드인지 확인 */
            if (direction == 1) /* 오른쪽 노드 */
                parentNode->right = childNode; /* 설정해
둔 자식노드로 대체 */
            else if (direction == -1) /* 왼쪽 노드 */
                parentNode->left = childNode; /* 설정해
둔 자식노드로 대체 */

            return 1;
        }
    }
}

```

```

/* 다음 자식 노드 탐색 이전에 현재 노드를 부모 노드로 설정 */
parentNode = p;

if (key > p->key)/* 찾는 키 값이 현재 노드보다 크다면 */
{
    direction = 1;/* 탐색할 자식노드의 방향이 오른쪽임을 기억한
다. */

    p = p->right;/* 오른쪽 노드 탐색 */
}
else /* 찾는 키 값이 현재 노드보다 작다면 */
{
    direction = -1;/* 탐색할 자식노드의 방향이 왼쪽임을 기억한
다. */

    p = p->left;/* 왼쪽 노드 탐색 */
}
}

/* 트리를 모두 탐색 하고도 삭제할 노드를 찾지 못했다면 */
printf("Cannot find the Node");/* 안내 출력 후 종료 */
return 0;
}

/* subtree 탐색 및 (최소값 키)노드 추출 함수 */
Node* minSub(Node* ptr)
{
    Node* temp;
    Node* parent = NULL;/* 서브트리가 하나의 노드만 존재하거나 왼쪽에 자식노
드가 없는 경우 parent는 NULL*/
    /* 키 값이 최소인 노드만 찾으면 되므로 왼쪽으로만 탐색한다.(이원 탐색 트리의
특성상 키값 비교 불필요) */
    while (ptr != NULL)/* 더이상 탐색할 노드가 없을 때 까지 반복*/
    {
        if (ptr->left == NULL)/* 왼쪽 자식 노드가 없다면 현재 노드가 최소
키값을 갖는 노드 */
        {
            if (parent != NULL)/* parent가 설정되었다면*/
            {
                parent->left = NULL;/* 부모 노드의 왼쪽 자식을 공
백처리 */
            }
            return ptr;/* 현재노드를 반환 */
        }
    }
}

```

```

        }
        else break; /* parent가 NULL이면 현재노드가 서브트리의
root노드이므로 break */
    }
    parent = ptr; /* 다음 노드 탐색 이전에 현재노드를 부모노드로 기억
한다. */
    ptr = ptr->left; /* 왼쪽 노드 탐색 */
}
return ptr; /* 삭제할 노드의 서브트리가 하나의 노드 뿐이거나, 왼쪽 노드가 없
는 경우 가장 작은 키값을 갖는 노드는 서브트리의 root노드 */
}

/* 메모리 해제 */
void freeNode(Node* ptr)
{
    if(ptr) { /* 현재노드가 공백이 아니면 */
        freeNode(ptr->left); /* 왼쪽 재귀(탐색 및 메모리 해제) */
        freeNode(ptr->right); /* 오른쪽 재귀(탐색 및 메모리 해제) */
        free(ptr); /* 메모리 해제 */
    }
}

int freeBST(Node* head)
{
    /* root노드가 head노드인 경우?? */
    if(head->left == head)
    {
        free(head);
        return 1;
    }

    Node* p = head->left; /* root노드 */

    freeNode(p); /* root노드(하위노드 포함) 메모리 해제 */
    free(head); /* head노드 메모리 해제 */

    return 1;
}

/* 스택 POP */

```

```

Node* pop()
{
    if (top == -1)/* 스택이 공백이면 */
    {
        /* 스택이 비어있는 경우 Iterative Inorder를 종료하므로 경고를 출력
할 필요가 없다. */
        /* printf("Stack is empty"); */
        return NULL; /* 종료 */
    }
    Node* temp = stack[top]; /* 스택에서 꺼낼 노드 저장 */
    stack[top] = NULL; /* 노드 삭제 */
    top--; /* 스택에 노드가 저장된 위치를 가르키는 top을 1감소시킨다 */

    return temp; /* 꺼낸 노드 temp를 반환 */
}

/* 스택 PUSH */
void push(Node* aNode)
{
    if (top == MAX_STACK_SIZE - 1)/* 스택이 가득 차있다면 */
    {
        /* 스택이 가득 차있더라도, Iterative Inorder 수행과정에서 pop된 이
후 다시 push하므로 경고를 출력할 필요가 없다. */
        /* printf("Stack is full"); */
        return; /* 종료 */
    }
    top++; /* top을 1만큼 증가시킨다 */
    stack[top] = aNode; /* 노드 삽입 */
}

/* 큐 deQueue */
Node* deQueue()
{
    /* 큐가 공백이면 종료 */
    if (front == rear) return NULL; /* front와 rear의 값이 같다면 큐에 어떤 노
드도 없다. */
    front++; /* 큐의 시작을 가르키는(front 방향으로 노드가 큐를 탈출(삭제)함)
front값을 1만큼 증가시킨다. */
    Node* temp = queue[front]; /* 삭제할 노드 저장 */
    queue[front] = NULL; /* 노드 삭제 */
}

```

```

        return temp; /* 노드 반환 */
    }

    /* 큐 enqueue */
    void enqueue(Node* aNode)
    {
        /* 큐가 가득 찼다면 */
        if (rear == MAX_QUEUE_SIZE - 1)
        {
            /* 선형 큐 앞으로 끌어당기기 */
            int size = rear - front; /* 큐에 남아있는 노드의 수를 구한다. */
            for (int i=0; i<size; i++) /* 남은 노드 수 만큼 반복 */
            {
                queue[i] = queue[front+(i+1)]; /* 노드들을 큐의 앞으로 이동
시킨다 */
                queue[front + (i + 1)] = NULL;
            }
            front = -1; /* front 초기화 */
            rear = size-1; /* rear값은 size-1로 초기화 하여 front와 rear간의
거리 유지 */
            return;
        }
        rear++; /* 큐의 끝을 가리키는 rear값을 1만큼 증가시킨다. */
        queue[rear] = aNode; /* 새로운 노드 삽입 */
    }
}

```

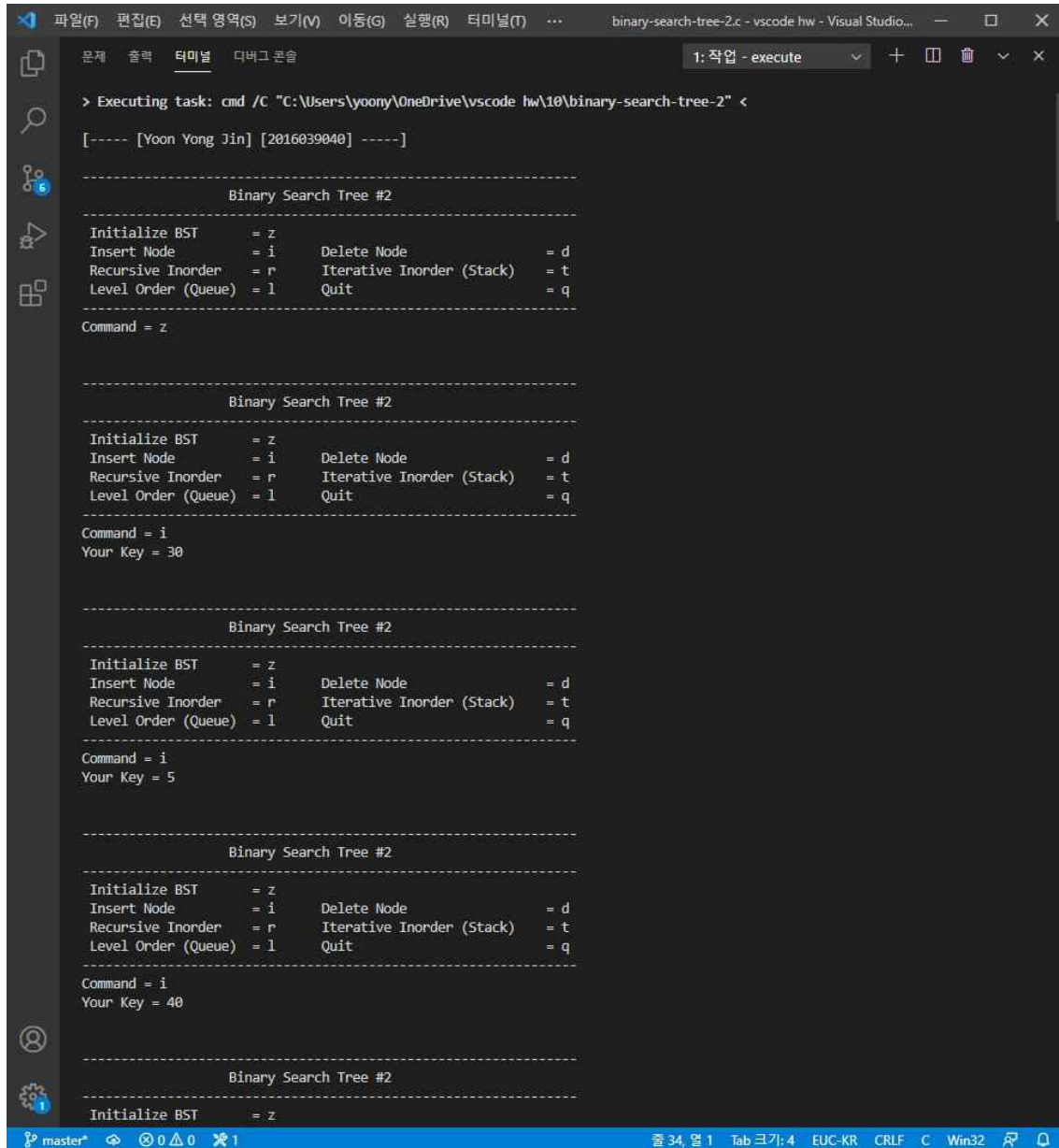
4. GitHub에 hw7 Repository를 생성하고 doubly-linked-list.c를 업로드 한다.

<https://github.com/uniz21/DataStructure-HW-10>



7. 보고서에 실행결과를 Screen Capture하여 첨부한다.

## 실행결과



```
binary-search-tree-2.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 1: 작업 - execute
> Executing task: cmd /C "C:\Users\yoony\OneDrive\vscode hw\10\binary-search-tree-2" <
[----- [Yoon Yong Jin] [2016039040] -----]

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = z

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = i
Your Key = 30

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = i
Your Key = 5

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = i
Your Key = 40

-----
Binary Search Tree #2
-----
Initialize BST      = z
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-2.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 1: 작업 - execute + - ✕
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder  = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = i
Your Key = 2

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder  = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = i
Your Key = 35

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder  = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = i
Your Key = 80

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder  = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = r
[2] [5] [30] [35] [40] [80]

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node        = i      Delete Node          = d
Recursive Inorder  = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit              = q
-----
Command = t
[2] [5] [30] [35] [40] [80]
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-2.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 1: 작업 - execute
-----
Initialize BST      = z
Insert Node         = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit           = q
-----
Command = 1
[30] [5] [40] [2] [35] [80]
-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit           = q
-----
Command = d
Your Key = 35
-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit           = q
-----
Command = 1
[30] [5] [40] [2] [80]
-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit           = q
-----
Command = d
Your Key = 5
-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node          = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit           = q
-----
Command = 1
[30] [2] [40] [80]
-----
master 0 0 1 줄 34, 열 1 Tab 크기: 4 EUC-KR CRLF C Win32
```

```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-2.c - vscode hw - Visual Studio...
문제 출력 터미널 디버그 콘솔 1: 작업 - execute
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = d
Your Key = 30

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = l
[40] [2] [80]

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = z

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 10

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 5

-----
Binary Search Tree #2
```

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

Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 20

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 3

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 8

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 18

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = i
Your Key = 25

master 0 0 0 1 줄 34, 열 1 Tab 크기: 4 EUC-KR CRLF C Win32
```

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

Insert Node      = i      Delete Node      = d
Recursive Inorder = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit          = q
-----
Command = i
Your Key = 7

-----
Binary Search Tree #2
-----
Initialize BST   = z
Insert Node      = i      Delete Node      = d
Recursive Inorder = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit          = q
-----
Command = i
Your Key = 23

-----
Binary Search Tree #2
-----
Initialize BST   = z
Insert Node      = i      Delete Node      = d
Recursive Inorder = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit          = q
-----
Command = i
Your Key = 30

-----
Binary Search Tree #2
-----
Initialize BST   = z
Insert Node      = i      Delete Node      = d
Recursive Inorder = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit          = q
-----
Command = i
Your Key = 21

-----
Binary Search Tree #2
-----
Initialize BST   = z
Insert Node      = i      Delete Node      = d
Recursive Inorder = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit          = q
-----
Command = i
Your Key = 24

master 0 1 줄 34, 열 1 Tab 크기: 4 EUC-KR CRLF C Win32
```

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

Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = l
[10] [5] [20] [3] [8] [18] [25] [7] [23] [30] [21] [24]

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = r
[3] [5] [7] [8] [10] [18] [20] [21] [23] [24] [25] [30]

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = d
Your Key = 3

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = l
[10] [5] [20] [8] [18] [25] [7] [23] [30] [21] [24]

-----
Binary Search Tree #2
-----
Initialize BST = z
Insert Node = i Delete Node = d
Recursive Inorder = r Iterative Inorder (Stack) = t
Level Order (Queue) = l Quit = q
-----
Command = d
Your Key = 8

-----
master 0 0 1 줄 34, 열 1 Tab 크기: 4 EUC-KR CRLF C Win32
```



```
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) ... binary-search-tree-2.c - vscode hw - Visual Studio...
문제 출력 터미널 디버거 콘솔 1: 작업 - execute + [] [] v x

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node           = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit                = q
-----
Command = d
Your Key = 8

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node           = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit                = q
-----
Command = l
[10] [5] [20] [7] [18] [25] [23] [30] [21] [24]

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node           = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit                = q
-----
Command = d
Your Key = 20

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node           = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit                = q
-----
Command = l
[10] [5] [21] [7] [18] [25] [23] [30] [24]

-----
Binary Search Tree #2
-----
Initialize BST      = z
Insert Node         = i      Delete Node           = d
Recursive Inorder   = r      Iterative Inorder (Stack) = t
Level Order (Queue) = l      Quit                = q
-----
Command =
```