『제 24기 청학동』Tutoring 6주 주간 학습 보고서

학습활동 참여 점검			
팀명	2+3=5		
1회차 일시	2022.04.26 18:00 ~ 21:00	1회차 장소	Discord
1회차 참석자	신수빈,변장무,이준호,이동현,권재현		
1회차 불참자			
2회차 일시		2회차 장소	
2회차 참석자			
2회차 불참자			
3회차 일시		3회차 장소	
3회차 참석자			
3회차 불참자			
4회차 일시		4회차 장소	
4회차 참석자			
4회차 불참자			
5회차 일시		5회차 장소	
5회차 참석자			
5회차 불참자			
학습목표	1. 트리, 2. 이진 트리, 3. 이진 탐색 트리		
과제물	이번주 과제물은 없다.		

『제 24기 청학동』Tutoring 6주 주간 학습 보고서

학습활동 참여 점검	
팀명	2+3=5

1. 트리 (Tree)

계층적인 자료를 표현하는데 적합한 자료구조이다.

부모–자식 관계의 노드 순서로 이루어지며 저장 순서가 중요하다.

응용분야

계층적인 조직 표현, 파일 시스템, 인공지능에서의 결정 트리

용어

노드 (node) : 트리의 구성요소

루트 (root) : 부모가 없는 노드(A)

서브 트리 (sub tree) : 하나의 노드와 그 노드들의 자손들로 이루어진 트리

단말 노드 (terminal node) : 자식이 없는 노드
비 단말 노드 : 적어도 하나의 자식을 가지는 내부 노드

* 자식, 부모, 형제, 조상, 자손 노드 -> 인간관계와 비슷한 개념

레벨 (level) : 트리의 각 레벨 번호

높이 (height) : 트리의 최대 레벨

차수 (degree) : 노드가 가지고 있는 자식 노드의 개수

2. 이진 트리 (Binary Tree)

모든 노드가 2개의 서브 트리를 가지고 있는 트리이다.

서브 트리는 자식 노드가 없는 공집합도 가능하다.

이진 트리에서 각 노드에 연결된 노드는 최대 2개의 자식 노드를 허용한다.

모든 노드의 차수가 2 이하로 구성되면 구현하기 편리해진다. (if 문으로 판별이 가능하다)

이진 검색 트리는 서브 트리의 크기 순서에 의해 좌우 링크로 연결 배치된다.

이진 트리의 성질

n 개의 노드를 가진 이진 트리는 정확하게 n-1개의 간선을 갖는다.

높이가 h인 이진 트리의 경우, 최소 h 개의 노드를 가지며 최대 2-1개의 노드를 갖는다.

n 개의 노드를 가지는 이진 트리의 높이는 최대 n이거나 최소 이 된다.

이진 트리의 분류

포화 이진 트리 (Full Binary Tree)
트리의 각 레벨에 노드가 꽉 차있는 형태의 이진 트리이다.
완전 이진 트리 (Complete Binary Tree)
높이가 h 일 때 레벨 1부터 h-1까지는 노드가 모두 채워져 있고, 마지막 레벨 h에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진 트리이다.

이진 트리를 구현하는 방법

배열 표현법
이진 트리를 완전 이진 트리라고 가정한다.
각 노드에 번호를 붙여서 그 번호를 배열의 첨자로 적용하여 노드를 배열에 저장하는 방법이다.

링크 표현법
데이터 필드와 2개의 포인터 필드를 가진 노드를 연결한 구조이다.
부모 노드보다 작으면 왼쪽 자식, 크면 오른쪽 자식의 주소로 저장된다.

```
#include

#include

#include


typedefstructTreeNode

{

    int data;

    struct TreeNode *left, *right;

} TreeNode;


intmain(void)

{

    TreeNode *n1, *n2, *n3;

    n1 = (TreeNode *)malloc(sizeof(TreeNode));

    n2 = (TreeNode *)malloc(sizeof(TreeNode));

    n3 = (TreeNode *)malloc(sizeof(TreeNode));

    n1->data = 10;

    n1->left = n2;

    n1->right = n3;

    n2->data = 20;

    n2->left = NULL;

    n2->right = NULL;

    n3->data = 30;

    n3->left = NULL;

    n3->right = NULL;

    free(n1); free(n2); free(n3);

    return 0;

}
```

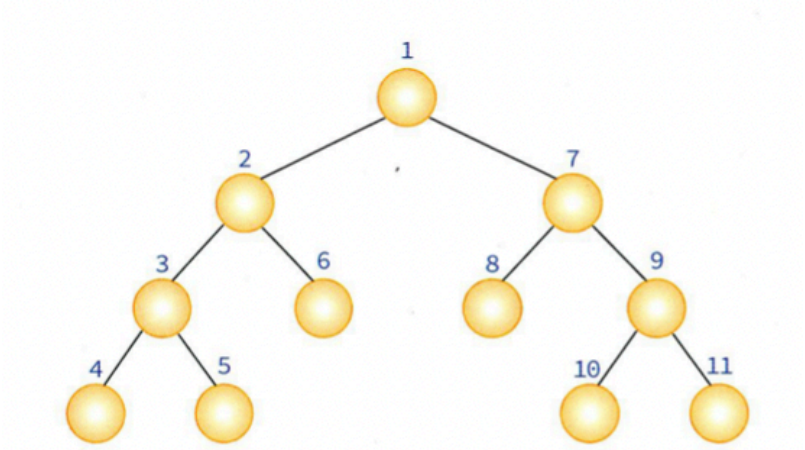
이진 트리의 순회

순회(traversal)

트리의 모든 노드들을 순서적으로 방문하는 동작이다.

전위 순회 (Pre-order Traversal)

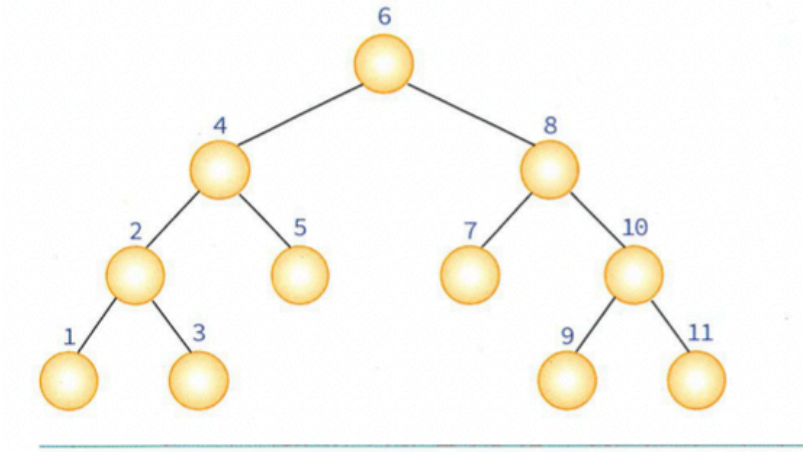
1. 루트 노드를 방문한다.
2. 왼쪽 서브 트리를 방문한다.
3. 오른쪽 서브 트리를 방문한다.



[그림 8-25] 전위순회에서의 노드 방문 순서

중위 순회 (In-order Traversal)

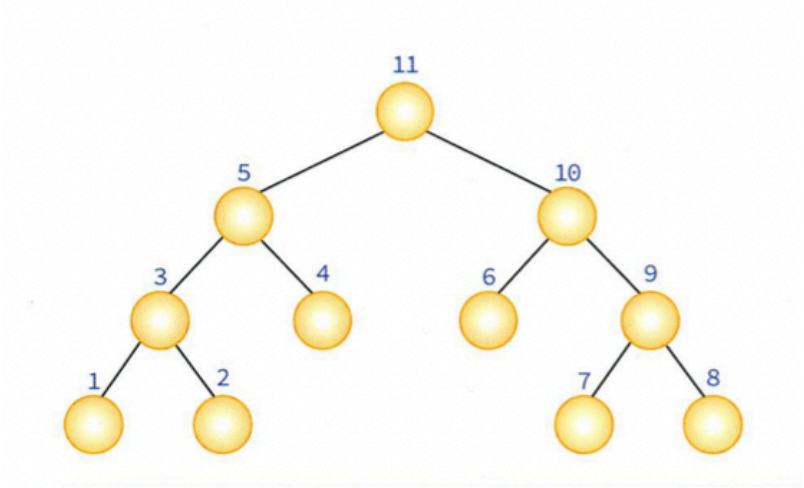
1. 왼쪽 서브 트리를 방문한다.
2. 루트 노드를 방문한다.
3. 오른쪽 서브 트리를 방문한다.



[그림 8-27] 중위 순회의 예

후위 순회 (Post-order Traversal)

1. 왼쪽 서브 트리의 모든 노드를 방문한다.
2. 오른쪽 서브 트리의 모든 노드를 방문한다.
3. 루트 노드를 방문한다.

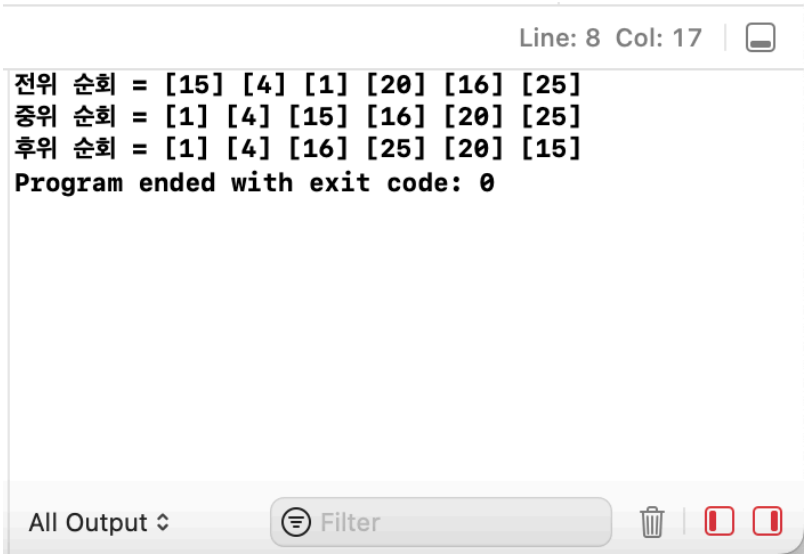


[그림 8-29] 후위 순회

```
#include
#include
#include

typedefstructTreeNode
{
    int data;
    struct TreeNode *left, *right;
} TreeNode;
//          15
//       4    20
//     1   16  25
TreeNode n1 = { 1, NULL, NULL };
TreeNode n2 = { 4, &n1, NULL };
TreeNode n3 = { 16, NULL, NULL };
TreeNode n4 = { 25, NULL, NULL };
TreeNode n5 = { 20, &n3, &n4 };
TreeNode n6 = { 15, &n2, &n5 };
TreeNode *root = &n6;

// 전위 순회
void preorderr(TreeNode *root)
{
    if (root != NULL)
    {
        printf("[%d] ", root->data);
        preorderr(root->left);
        preorderr(root->right);
```

```
    }
}

// 중위 순회
void inorder(TreeNode *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("[%d] ", root->data);
        inorder(root->right);
    }
}

// 후위 순회
voidpostorderr(TreeNode *root)
{
    if (root != NULL)
    {
        postorderr(root->left);
        postorderr(root->right);
        printf("[%d] ", root->data);
    }
}

intmain(void)
{
    printf("전위 순회 = ");
    preorderr(root);
    printf("₩n");

    printf("중위 순회 = ");
    inorder(root);
    printf("₩n");

    printf("후위 순회 = ");
    postorderr(root);
    printf("₩n");
```

```
    return 0;

}
```



반복적 순회

```
#include
#include
#include

typedefstructTreeNode
{
    int data;
    struct TreeNode *left, *right;
} TreeNode;

#define SIZE 100
int top = -1;
TreeNode *stack[SIZE];

voidpush(TreeNode *p)
{
    if (top < SIZE - 1)
        stack[++top] = p;
}

TreeNode *pop()
{
    TreeNode *p = NULL;
    if (top >= 0)
        p = stack[top--];
    return p;
}

voidinorder_iter(TreeNode *root)
{
    while (1)
    {
        for (; root; root = root->left)
            push(root);
        root = pop();
        if (!root) break;
        printf("[%d] ", root->data);
        root = root->right;
    }
}

//         15
//       4    20
```

```
//      1    16  25

TreeNode n1 = { 1,   NULL, NULL };
TreeNode n2 = { 4,   &n1,   NULL };
TreeNode n3 = { 16, NULL, NULL };
TreeNode n4 = { 25, NULL, NULL };
TreeNode n5 = { 20, &n3,   &n4 };
TreeNode n6 = { 15, &n2,   &n5 };
TreeNode *root = &n6;

intmain(void)
{
    printf("중위 순회 = ");
    inorder_iter(root);
    printf("₩n");
    return 0;
}
```

Line: 44 Col: 20

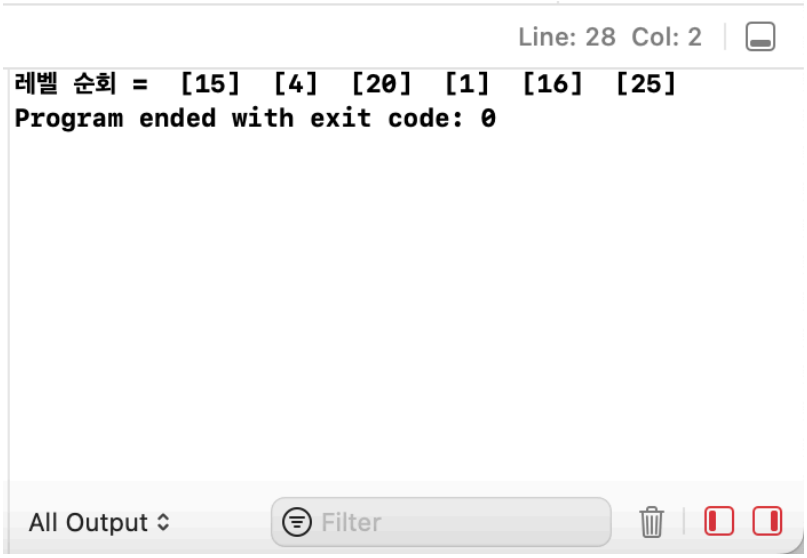
```
중위 순회 = [1] [4] [15] [16] [20] [25]
Program ended with exit code: 0
```

All Output Filter

레벨 순회

```
#include
#include
#include

typedefstructTreeNode
{
    int data;
    struct TreeNode *left, *right;
} TreeNode;

#define MAX_QUEUE_SIZE 100
typedefTreeNode * element;
typedefstruct
{
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

void error(char *message)
{
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

voidinit_queue(QueueType *q)
{
    q->front = q->rear = 0;
}

intis_empty(QueueType *q)
{
    return (q->front == q->rear);
```

```
}

intis_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

voidenqueue(QueueType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태입니다.");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}

elementdequeue(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다.");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

voidlevel_order(TreeNode *ptr)
{
    QueueType q;

    init_queue(&q);

    if (ptr == NULL) return;
    enqueue(&q, ptr);
    while (!is_empty(&q)) {
        ptr = dequeue(&q);
        printf(" [%d] ", ptr->data);
        if (ptr->left)
            enqueue(&q, ptr->left);
        if (ptr->right)
            enqueue(&q, ptr->right);
    }
}

//          15
//        4    20
//      1    16  25

TreeNode n1 = { 1,  NULL, NULL };
TreeNode n2 = { 4,  &n1,  NULL };
TreeNode n3 = { 16, NULL, NULL };
TreeNode n4 = { 25, NULL, NULL };
TreeNode n5 = { 20, &n3,  &n4 };
TreeNode n6 = { 15, &n2,  &n5 };
TreeNode *root = &n6;

intmain(void)
{
    printf("레벨 순회 = ");
    level_order(root);
    printf("₩n");
    return 0;
}
```



이진 트리의 응용

수식 트리 처리

수식 트리는 산술연산자와 피연산자로 구성되며, 피연산자들은 단말 노드가 되고 연산자는 비단말 노드가 된다.

수식 트리 계산 알고리즘

1. 만약 수식 트리가 공백 상태이면 그냥 복귀한다.
2. 수식 트리가 공백 상태가 아니면 왼쪽 서브 트리를 계산하기 위하여 다시 순환 호출한다. 이때 매개변수는 왼쪽 자식 노드가 된다.
3. 똑같은 식으로 오른쪽 서브 트리를 계산한다.
4. 루트 노드의 데이터 필드에서 연산자를 추출한다.
5. 추출된 연산자를 가지고 연산을 수행해서 반환한다.

```
#include

#include


typedefstructTreeNode

{

    int data;

    struct TreeNode *left, *right;

} TreeNode;


//          +

//      *       +

//    1   4   16  25


TreeNode n1 = { 1,   NULL, NULL };

TreeNode n2 = { 4,   NULL,  NULL };

TreeNode n3 = { '*',  &n1,  &n2 };

TreeNode n4 = { 16, NULL, NULL };

TreeNode n5 = { 25, NULL, NULL };

TreeNode n6 = { '+', &n4,  &n5 };

TreeNode n7 = { '+', &n3,  &n6 };
```

학습 내용

```
TreeNode *expp = &n7;

int evaluate(TreeNode *root)
{
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return root->data;
    else {
        int op1 = evaluate(root->left);
        int op2 = evaluate(root->right);
        printf("%d %c %d을 계산합니다.₩n", op1, root->data, op2);
        switch (root->data) {
        case '+':
            return op1 + op2;
        case '-':
            return op1 - op2;
        case '*':
            return op1 * op2;
        case '/':
            return op1 / op2;
        }
    }
    return 0;
}
//
intmain(void)
{
    printf("수식의 값은 %d입니다. ₩n", evaluate(expp));
    return 0;
}
```

Line: 24 Col: 2

```
1 * 4을 계산합니다.
16 + 25을 계산합니다.
4 + 41을 계산합니다.
수식의 값은 45입니다.
Program ended with exit code: 0
```

All Output Filter

디렉토리 용량 계산

```
#include
#include

typedefstructTreeNode
{
    int data;
    struct TreeNode *left, *right;
} TreeNode;

intcalc_dir_size(TreeNode *root)
{
    int left_size, right_size;
    if (root == NULL) return 0;

    left_size = calc_dir_size(root->left);
    right_size = calc_dir_size(root->right);
    return (root->data + left_size + right_size);
}
//
intmain(void)
{
    TreeNode n4 = { 500, NULL, NULL };
    TreeNode n5 = { 200, NULL, NULL };
    TreeNode n3 = { 100, &n4, &n5 };
    TreeNode n2 = { 50, NULL, NULL };
    TreeNode n1 = { 0, &n2, &n3 };

    printf("디렉토리의 크기 = %d₩n", calc_dir_size(&n1));
}
```

Line: 22 Col: 39

```
디렉토리의 크기 = 850
Program ended with exit code: 0
```

All Output Filter

이진 트리의 추가 연산

노드 개수 계산
이진 트리에 저장된 노드 개수를 모두 계산한다.

개념
부모 노드의 왼쪽, 오른쪽 서브 트리를 각각 순환 호출하여 서브 트리의 노드 개수를 계산한 후, 부모 노드에게 개수를 반환한다.
순환 과정
총개수 = 부모 노드 1개 + 왼쪽 서브 트리 개수 + 오른쪽 서브 트리 개수 -> 다시 부모 노드에게 반환 -> 루트에서도 동일한 과정으로 계산하여 전체 개수를 카운팅 한다.

높이 계산
이진 트리의 높이를 계산하기 위하여 왼쪽, 오른쪽 서브 트리에 대하여 각각 순환 호출하면 서브 트리의 높이가 반환되므로 그중에서 최댓값을 구하여 반환하면 된다.

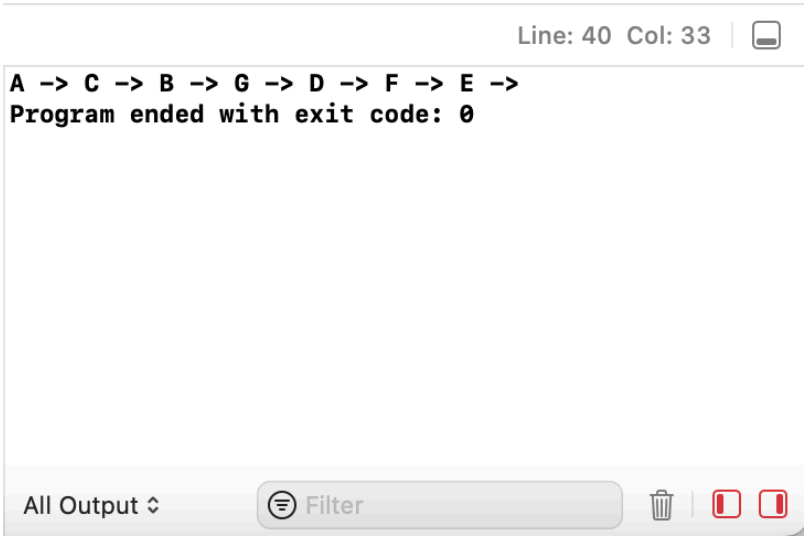
스레드 이진 트리

트리는 단일 방향 구조이므로 순회하려면 함수 호출/리턴의 순환 동작이 기본이다.
이때 단말 노드는 모두 NULL 링크이므로 NULL 링크를 활용하면 함수 호출에 의한 트리 순회를 하지 않고도 노드들을 쉽게 순회할 수 있는 방법이 가능하다.
- 단말 노드의 NULL 링크에 중위 순회의 다음 순서 노드에 대한 주소(포인터)를 저장한다.
- 즉, 중위 순회에서 다음 순서의 노드 주소를 연속적으로 저장한다면 함수 호출이 불필요하다.
-> 자식 노드로 이동은 llink/rlink를 이용하고, 부모 노드로 이동은 스레드 link 이용한다.

단말/비단말 노드 구별을 위해 is_thread 필드를 포함시켜 노드 구조를 재구성한다.

```
#include

#define TRUE 1

#define FALSE 0


typedefstructTreeNode

{

    int data;

    struct TreeNode *left, *right;

    int is_thread;

} TreeNode;


//          G

//      C       F

//    A   B   D   E


TreeNode n1 = { 'A',  NULL, NULL, 1 };

TreeNode n2 = { 'B',  NULL,  NULL, 1 };

TreeNode n3 = { 'C',  &n1,  &n2, 0 };

TreeNode n4 = { 'D', NULL, NULL, 1 };

TreeNode n5 = { 'E', NULL, NULL, 0 };
```

```
TreeNode n6 = { 'F', &n4,  &n5, 0 };
TreeNode n7 = { 'G', &n3,  &n6, 0 };
TreeNode * expp = &n7;

TreeNode * find_successor(TreeNode * p)
{
    TreeNode * q = p->right;
    if (q == NULL || p->is_thread == TRUE)
        return q;

    while (q->left != NULL) q = q->left;
    return q;
}

voidthread_inorder(TreeNode * t)
{
    TreeNode * q;

    q = t;
    while (q->left) q = q->left;
    do
    {
        printf("%c -> ", q->data);
        q = find_successor(q);
    } while (q);
}

intmain(void)
{
    n1.right = &n3;
    n2.right = &n7;
    n4.right = &n6;

    thread_inorder(expp);
    printf("₩n");
    return 0;
}
```



3. 이진 탐색 트리 (Binary Search Tree)

이진 트리 기반의 탐색을 위한 자료구조로 트리에서 가장 많이 사용하는 중요한 자료구조이다.

이진 탐색 트리의 정의

모든 원소의 키는 유일한 키를 가진다.

왼쪽 서브 트리 키들은 루트 키보다 작다.

오른쪽 서브 트리의 키들은 루트의 키보다 크다.

왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.

이진 (탐색) 트리로 구성하는 목적은 매우 빠르고 효율적인 탐색을 위해서이다.

순환적인 탐색 연산

이진 탐색 트리에서 특정한 키값을 가진 노드를 찾기 위해서는 탐색 키값과 루트 노드의 키값을 비교한다.
1. 탐색 키값 = 루트 노드의 키값
   탐색이 성공적으로 끝나고 트리의 노드 주소를 리턴한다.
2. 탐색 키값 < 루트 노드의 키값
   루트 노드의 왼쪽 자식을 기준으로 탐색을 다시 시작한다.
3. 탐색 키값 > 루트 노드의 키값
   루트 노드의 오른쪽 자식을 기준으로 탐색을 다시 시작한다.

이진 탐색 트리에서의 삽입 연산

이진 탐색 트리에 원소를 삽입하기 위해서는 먼저 삽입 위치를 탐색해야 한다.

탐색을 성공하면 이미 해당 원소가 트리 안에 존재하는 것이므로, 키가 중복되기 때문에 삽입이 불가능하다.

탐색을 실패하면 실패한 위치가 바로 새로운 노드를 삽입하는 위치가 된다.

이진 탐색 트리에서의 삭제 연산

1. 삭제하려는 노드가 0개의 자식 노드를 갖는 단말 노드인 경우

: 단말 노드의 부모 노드를 찾아서 연결을 끊으면 된다.

2. 삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리 중 1개의 자식 노드만 가지고 있는 경우

: 해당 노드는 삭제하고 서브 트리를 부모 노드에 붙여준다.

즉, 부모 노드에서 해당 노드를 제치고 서브 트리로 바로 연결하는 방식을 취한다.

3. 삭제하려는 노드가 2개의 서브 트리를 모두 가지고 있는 경우

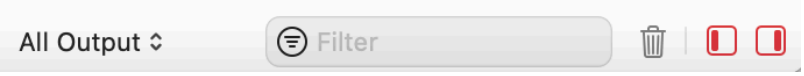
: 삭제할 노드와 가장 가까운 크기의 값을 가진 노드를 삭제 노드 위치로 가져온다.

즉, 2개의 서브 트리 중에서 해당 노드를 대신할 노드를 선별해서 대체한다.

```
#include
#include

typedefintelement;
typedefstructTreeNode
{
    element key;
    struct TreeNode *left, *right;
} TreeNode;

TreeNode * search(TreeNode * node, int key)
{
    if (node == NULL) return NULL;
    if (key == node->key) return node;
    else if (key < node->key)
        return search(node->left, key);
    else
        return search(node->right, key);
}

TreeNode * new_node(int item)
{
    TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

TreeNode * insert_node(TreeNode * node, int key)
{
    if (node == NULL) return new_node(key);

    if (key < node->key)
        node->left = insert_node(node->left, key);
    else if (key > node->key)
        node->right = insert_node(node->right, key);

    return node;
}

TreeNode * min_value_node(TreeNode * node)
{
    TreeNode * current = node;

    while (current->left != NULL)
        current = current->left;

    return current;
}
```

```c
TreeNode * delete_node(TreeNode * root, int key)
{
    if (root == NULL) return root;

    if (key < root->key)
        root->left = delete_node(root->left, key);

    else if (key > root->key)
        root->right = delete_node(root->right, key);

    else
    {
        if (root->left == NULL)
        {
            TreeNode * temp = root->right;
            free(root);
            return temp;
        }

        else if (root->right == NULL)
        {
            TreeNode * temp = root->left;
            free(root);
            return temp;
        }

        TreeNode * temp = min_value_node(root->right);

        root->key = temp->key;
        root->right = delete_node(root->right, temp->key);
    }
    return root;
}

void inorder(TreeNode * root)
{
    if (root)
    {
        inorder(root->left);
        printf("[%d] ", root->key);
        inorder(root->right);
    }
}

intmain(void)
{
    TreeNode * root = NULL;
    TreeNode * tmp = NULL;

    root = insert_node(root, 30);
    root = insert_node(root, 20);
    root = insert_node(root, 10);
    root = insert_node(root, 40);
    root = insert_node(root, 50);
    root = insert_node(root, 60);

    printf("이진 탐색 트리 중위 순회 결과 ₩n");
    inorder(root);
    printf("₩n₩n");
    if (search(root, 30) != NULL)
        printf("이진 탐색 트리에서 30을 발견함 ₩n");
    else
        printf("이진 탐색 트리에서 30을 발견 못함₩n");
    return 0;
}
```

```
Line: 14  Col: 15
이진 탐색 트리 중위 순회 결과
[10] [20] [30] [40] [50] [60]

이진 탐색 트리에서 30을 발견함
Program ended with exit code: 0
```



이진 탐색 트리의 성능

이진 탐색 트리에서의 탐색, 삽입, 삭제 연산의 시간 복잡도는 트리 높이에 비례한다.

최선의 경우

- 이진 트리가 균형적으로 생성된 경우이다.
- 높이 h(n) = logn
- 이진 트리로 탐색하는 성능이 우수하다.

최악의 경우

- 한 쪽으로 치우친 경사진 이진 트리의 경우이다.
- 높이 h(n) = n
- 이진 트리로 탐색하는 성능이 순차 탐색과 같아진다. (이진 탐색 트리에서는 최악의 상태)

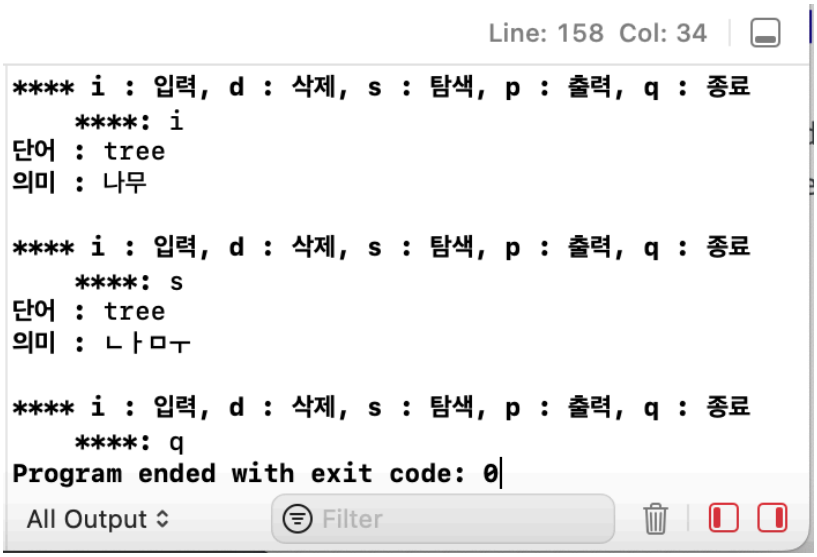
이진 탐색 트리의 응용

영어 사전

```
#include
#include
#include
#include

#define MAX_WORD_SIZE 100
#define MAX_MEANING_SIZE 200

typedefstruct {
    char word[MAX_WORD_SIZE];
    char meaning[MAX_MEANING_SIZE];
} element;

typedefstructTreeNode
{
    element key;
    struct TreeNode *left, *right;
} TreeNode;

intcompare(element e1, element e2)
{
    return strcmp(e1.word, e2.word);
```

```
}

voiddisplay(TreeNode * p)
{
    if (p != NULL) {
        printf("(");
        display(p->left);
        printf("%s:%s", p->key.word, p->key.meaning);
        display(p->right);
        printf(")");
    }
}

TreeNode * search(TreeNode * root, element key)
{
    TreeNode * p = root;
    while (p != NULL)
    {
        if (compare(key, p->key) == 0)
            return p;
        else if (compare(key, p->key) < 0)
            p = p->left;
        else if (compare(key, p->key) > 0)
            p = p->right;
    }
    return p;
}

TreeNode * new_node(element item)
{
    TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

TreeNode * insert_node(TreeNode * node, element key)
{
    if (node == NULL) return new_node(key);

    if (compare(key, node->key)<0)
        node->left = insert_node(node->left, key);

    else if (compare(key, node->key)>0)
        node->right = insert_node(node->right, key);

    return node;
}

TreeNode * min_value_node(TreeNode * node)
{
    TreeNode * current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}

TreeNode * delete_node(TreeNode * root, element key)
{
    if (root == NULL) return root;

    if (compare(key, root->key)<0)
        root->left = delete_node(root->left, key);

    if (compare(key, root->key)>0)
        root->right = delete_node(root->right, key);

    else
    {
        if (root->left == NULL)
        {
```

```
            TreeNode * temp = root->right;
            free(root);
            return temp;
        }

        else if (root->right == NULL)
        {
            TreeNode * temp = root->left;
            free(root);
            return temp;
        }

        TreeNode * temp = min_value_node(root->right);

        root->key = temp->key;

        root->right = delete_node(root->right, temp->key);
    }
    return root;
}

voidhelp()
{
    printf("₩n**** i : 입력, d : 삭제, s : 탐색, p : 출력, q : 종료 ****: ");
}

intmain(void)
{
    char command;
    element e;
    TreeNode * root = NULL;
    TreeNode * tmp;

    do
    {
        help();
        command = getchar();
        getchar();
        switch (command)
        {
            case 'i':
                printf("단어 : ");
                gets(e.word);
                printf("의미 : ");
                gets(e.meaning);
                root = insert_node(root, e);
                break;

            case 'd':
                printf("단어 : ");
                gets(e.word);
                root=delete_node(root, e);
                break;

            case 'p':
                display(root);
                printf("₩n");
                break;

            case 's':
                printf("단어 : ");
                gets(e.word);
                tmp = search(root, e);
                if (tmp != NULL)
                    printf("의미 : %s₩n", e.meaning);
                break;
        }
    } while (command != 'q');

    return 0;
}
```


Line: 158 Col: 34
**** i : 입력, d : 삭제, s : 탐색, p : 출력, q : 종료
****: i
단어 : tree
의미 : 나무
**** i : 입력, d : 삭제, s : 탐색, p : 출력, q : 종료
****: s
단어 : tree
의미 : ㄴㅏㅁㅜ
**** i : 입력, d : 삭제, s : 탐색, p : 출력, q : 종료
****: q
Program ended with exit code: 0
All Output
Filter

『제 24기 청학동』Tutoring 6주 주간 학습 보고서

학습활동 참여 점검	
팀명	2+3=5
성찰활동 잘한 점	신수빈 : 학교의 중간고사와 자격증 시험, 청학동 튜터링 수업 준비가 합쳐져 포기해버리고 싶었지만 시간을 세밀하게 계획해서 모두 차질 없도록 진행했다.변장무 : 트리구조에 대해 관심이 평소에 많이 있었다. 오늘 활동으로 각 트리의 개념 등을 자세하게 알게 되어 스스로 좋았고 관심 분야에 대해 알게 되어 뿌듯해하는 나 자신이 대견하다고 생각했다.이동현 : 시험 기간임에도 불구하고 모두가 집중해서 잘 했다.이준호 : 다른 친구들이 코드에서 찾지 못한 오류가 나는 부분을 도와주었다.권재현 : 자료구조 강의 보다 현재 청학동 튜터링의 진도가 더 빠르기 때문에 서버 구축 강의에서 배웠던 트리 구조를 떠올리면서 문제에 적용시켰다.
성찰활동 개선할 점	신수빈 : 청학동 특성상 계획된 일자, 시간에 활동을 진행해야 하는데 정해진 8주 내에 중간고사 기간이 끼어 있어 일정이 벅찬 감이 없지 않아 있었다.변장무 : 평소에도 내가 관심 있는 분야에 대해서만 이러한 집중력과 관심을 가지는 것에 대해, 조금 덜 흥미가 있거나 조금 덜 관심이 있는 분야에 대해서도 이러한 집중력을 보이도록 노력해야겠다고 생각했다.이동현 : 시험 공부 할 시간이 부족해졌다.이준호 : 없다.권재현 : 시험 기간에 진행된 수업이라서 다른 과목의 시험 생각이 계속 났다.
구성원 학습활동 사진	
다음 학습할 내용	[정렬] 1. 선택 정렬, 2. 삽입 정렬, 3. 버블 정렬, 4. 쉘 정렬, 5. 합병 정렬, 6. 히프 정렬