

# 자료구조2 실습

## 4주차 과제



제출일	21.09.27	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

# | 목 차 |

## 1. 이진 트리 1

- 1.1 문제 분석
- 1.2 소스 코드
- 1.3 소스 코드 분석
- 1.4 실행 창
- 1.5 느낀 점

## 2. 이진 트리 2

- 2.1 문제 분석
- 2.2 소스 코드
- 2.3 소스 코드 분석
- 2.4 실행 창
- 2.5 느낀 점

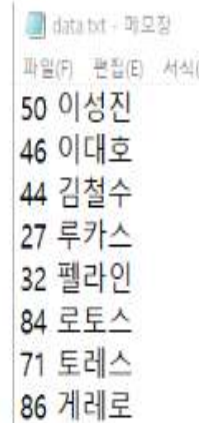
## 3. 이진 트리를 하며 느낀 점

## 1. 이진 트리 1

### 이진트리 삽입과 전위, 중위, 후위 출력

- 파일 data.txt에 오른쪽과 같은 학번과 이름들이 저장되어 있다.

data.txt파일에서 데이터를 입력받고 학번을 이용하여 이진트리에 저장 후 저장된 정보를 전위, 중위, 후위 순으로 출력한다.



```
data.txt - 메모장
파일(F) 편집(E) 서식(S)
50 이성진
46 이대호
44 김철수
27 루카스
32 펠라인
84 로토스
71 토레스
86 게레로
```

#### 1.1 문제 분석

조건 1 data.txt파일에서 데이터를 입력

조건 2 학번을 이용하여 insert\_node함수로 이진트리에 저장

조건 3 저장된 정보를 전위, 중위, 후위 순으로 출력

- 이 문제는 링크를 활용한 이진 트리 알고리즘으로, 트리는 왼쪽 또는 오른쪽을 가리키는 링크로 구성된다.

이번 문제의 경우 학번을 이용하여 이진트리에 저장해야 하므로 만약 부모 노드의 학번보다 학번이 작다면 왼쪽에 저장되고, 더 크다면 오른쪽에 저장되어야 할 것이다. 파일에 저장되어있는 순서대로 트리에 저장하고, 이름은 메모리를 아끼기 위해 포인터로 선언하고 동적할당을 이용하여 저장한다.

출력은 전위, 중위, 후위 순으로 하는데, 번호, 이름순으로 출력 후, 다음 노드를 가리키는 화살표를 같이 출력해야한다. 하지만 제일 마지막 노드에서는 화살표가 출력되면 안 되므로 이 부분을 고려하여 프로그램을 짜야 할 것이다.

## 1.2 소스 코드

[illegible]

## 1.3 소스 코드 분석

---

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 이진트리에 파일을 저장하고
            전위, 중위, 후위순으로 출력하는 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Student{
    int number;
    char *name;
}Student;

typedef struct TreeNode {
    Student data;
    struct TreeNode *left, *right;
}TreeNode;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다.
2. typedef를 이용하여 노드에 들어갈 요소의 자료형을 생성한다. 학생 구조체를 선언한다. 구조체 안에는 학번과 이름이 존재한다.
3. 학생이름의 경우 메모리 낭비를 줄이기 위해 포인터로 선언하였고, 이름에 맞는 길이로 동적할당하여 삽입할 수 있도록 한다.
4. 각 항을 하나의 노드로 표현한다. 노드에는 Student 자료형 변수 하나와 왼쪽 노드와 오른쪽 노드를 가리키는 left, right링크가 있다.

```

TreeNode* Insert_node(TreeNode*, Student);
TreeNode* new_node(Player);
void Print_Order(TreeNode*);
void Preorder(TreeNode*, int);
void Inorder(TreeNode*, int);
void Postorder(TreeNode*, int);
int Get_Node_Count(TreeNode*);
void Delete_Tree(TreeNode*);

```

## 6. 필요한 함수들을 선언한다.

- Insert\_node함수는 이진탐색트리에 삽입을 하는 함수이다.
- new\_node함수는 동적으로 노드를 할당하여 새로운 노드를 반환하는 함수이다
- Print\_Order함수는 이진트리를 전위, 중위, 후위 순으로 출력하는 함수이다.
- Preorder함수는 트리를 전위 순회하며 출력하는 함수이다.
- Inorder함수는 트리를 중위 순회하며 출력하는 함수이다.
- Postorder함수는 트리를 후위 순회하며 출력하는 함수이다.
- Get\_Node\_Count함수는 노드의 개수를 구하여 반환하는 함수이다.
- Delete\_Tree함수는 트리 안에 있는 노드 전체를 삭제하는 함수이다.

```

int main() {
    FILE *fp;
    char s[10];
    Student temp;
    TreeNode* root = NULL;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open\n");
        return 0;
    }
    while (!feof(fp)) {
        fscanf(fp, "%d%s", &temp.number, s);
        temp.name = (char*)malloc(sizeof(char)*(strlen(s) + 1));
        strcpy(temp.name, s);
        root = Insert_node(root, temp);
    }
    Print_Order(root);

    Delete_Tree(root);
    fclose(fp);
    return 0;
}

```

## 7. 필요한 변수들을 선언한다.

- fp는 파일포인터, s는 문자열을 입력받을 임시변수이다.
- temp는 삽입하기 전 입력받을 임시변수이고 root는 이진트리를 나타낸다.

8. 임시변수 s에 문자열을 입력받고, temp의 name을 s문자열의 길이에 \0문자를 포함하여 +1하러 동적할당한 후 문자열 복사함수 strcpy를 통해 문자열을 복사한다.

9. temp에 모든 값이 입력되었다면 이진트리에 삽입한다. insert\_node함수는 새로운 노드 또는 변경된 루트 포인터를 반환하므로 root변수에 이 값을 받아주도록 한다.

10. 모든 노드가 입력되었다면, Print\_Order함수를 호출한다. 이 함수는 이진트리를 전위, 중위, 후위의 순서로 출력을 하는 함수이다.  
가독성을 위하여 메인함수를 최대한 간단하게 프로그래밍 하였다.

11. 출력이 끝나면 노드 전체를 삭제하는 Delete\_Tree함수를 호출한다.  
이 함수는 이진트리에 있는 노드 전체를 삭제하는 함수이다.  
fclose를 이용해 파일을 닫고 파일을 종료한다.

```
TreeNode* Insert_node(TreeNode* node, Student key) {
    if (node == NULL) return new_node(key);

    if (key.number < node->data.number)
        node->left = Insert_node(node->left, key);
    else if (key.number > node->data.number)
        node->right = Insert_node(node->right, key);
    else {
        printf("번호가 중복됩니다. (번호 : %d, 이름 : %s) 삽입 불가\n", key.number, key.name);
    }
    return node;
}

TreeNode* new_node(Student item) {
    TreeNode* temp = (TreeNode*)malloc(sizeof(TreeNode));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}
```

12. Insert\_node함수는 이진트리에 노드를 삽입하는 함수이다. 매개 변수로는 트리와 삽입할 key값을 전달받는다.

13. 만약 트리가 공백이면 새로운 노드를 반환한다. 트리가 공백이 아니면 순환적으로 트리를 내려가는데, 만약 key학번이 노드 학번보다 작다면 왼쪽으로 내려가고 크다면 오른쪽으로 내려간다.  
새로운 노드 또는 변경된 루트포인터를 left또는 right에 입력받는다.

14. 만약 중복된 번호를 입력받게 된다면 이를 알리는 메시지를 출력한다.  
이름은 동명이인이 있을 수도 있기 때문에 번호만 가지고 판단하였다.

번호가 중복됩니다. (번호 : 50, 이름 : aaa) 삽입 불가  
마지막으로 변경된 루트포인트를 반환한다.

15. new\_node함수는 동적으로 메모리를 할당하여 새로운 노드를 생성 후 반환하는 함수이다.

임시 노드 temp에 값을 넣고 left와 right값을 NULL로 설정한 다음 이를 리턴한다.

```
void Print_Order(TreeNode*root) {
    int node_count = Get_Node_Count(root);
    printf("전위 순회 : ");
    Preorder(root, node_count);
    printf("\n중위 순회 : ");
    Inorder(root, node_count);
    printf("\n후위 순회 : ");
    Postorder(root, node_count);
}

int Get_Node_Count(TreeNode*node) {
    int count = 0;
    if (node != NULL)
        count = 1 + Get_Node_Count(node->left) + Get_Node_Count(node->right);

    return count;
}
```

16. Print\_Order함수는 이진트리를 전위, 중위, 후위 순으로 출력하는 함수이다. 먼저 노드의 개수를 node\_count변수에 저장한다.

17. 먼저 전위 순회라는 메시지를 출력하고 Preorder함수로 전위 순회 출력한다. 매개 변수로는 이진트리와 노드 개수를 전달한다.

중위와 후위도 위와 똑같이 진행한다.

18. Get\_Node\_Count함수는 이진트리의 노드 개수를 리턴하는 함수이다.

노드의 개수를 세기 위해서는 트리안의 노드들을 전체적으로 순회한다.

각각의 서브트리에 대하여 순환 호출한 다음, 반환되는 값에 1을 더하여 반환한다.



```

void Preorder(TreeNode* node, int count) {
    static int num = 0;
    if (node != NULL) {
        printf("%d %s ", node->data.number, node->data.name);
        num++;
        if (num < count) printf("-> ");
        Preorder(node->left, count);
        Preorder(node->right, count);
    }
}

void Inorder(TreeNode* node, int count) {
    static int num = 0;
    if (node != NULL) {
        Inorder(node->left, count);
        printf("%d %s ", node->data.number, node->data.name);
        num++;
        if (num < count) printf("-> ");
        Inorder(node->right, count);
    }
}

void Postorder(TreeNode* node, int count) {
    static int num = 0;
    if (node != NULL) {
        Postorder(node->left, count);
        Postorder(node->right, count);
        printf("%d %s ", node->data.number, node->data.name);
        num++;
        if (num < count) printf("-> ");
    }
}

```

19. Preorder(전위), Inorder(중위), Postorder(후위) 함수 모두 비슷한 알고리즘으로 순환을 이용해 출력한다.

출력을 할 때 제일 마지막 노드는 ->표시가 출력되면 안 되므로 정적 변수를 이용하여 노드의 개수와 이를 비교하며 출력하였다.

20. 전위함수는 먼저 현재 노드를 출력하고 왼쪽 노드 출력 후 오른쪽 노드를 출력한다. 정적 변수 num은 함수를 돌 때마다 1씩 증가하는데, 만약 노드의 개수 count와 개수가 같다면 제일 마지막 노드임을 알 수 있다. 따라서 num이 count보다 작을 때만 ->표시가 출력되도록 코딩하였다.

21. 중위함수도 비슷한 알고리즘으로 진행되는데, 중위는 먼저 가장 왼쪽 노드를 출력한 후 부모노드, 오른쪽 노드 순으로 출력이 된다. 후위함수는 아래의 왼쪽, 오른쪽 노드를 출력한 후 부모노드를 출력한다.

```

void Delete_Tree(TreeNode*node) {
    if (node != NULL) {
        Delete_Tree(node->left);
        Delete_Tree(node->right);
        printf("#\n삭제 한 노드 : %d, %s", node->data.number, node->data.name);

        free(node);
    }
}

```

22. Delete\_Tree함수는 이진트리 전체를 삭제하는 함수이다.

23. 부모 노드를 먼저 삭제하게 되면 그 밑의 노드들은 접근할 수 없기 때문에 가장 아래에 있는 단말 노드부터 삭제한다. 따라서 왼쪽 자식노드, 오른쪽 자식노드를 삭제하고 부모 노드를 삭제한다.

24. 어떤 노드가 삭제되는지 출력하고 free로 노드를 삭제한다.

후위알고리즘과 똑같이 했기 때문에 결과적으로 루트노드가 마지막으로 삭제될 것이고 모든 노드가 메모리 해제될 것이다.

## 1.4 실행 창

### - 문제 data파일대로 출력과 삭제된 노드 출력

```

전위 순회 : 50 이성진 -> 46 이대호 -> 44 김철수 -> 27 루카스 -> 32 펠라인 -> 84 로토스 -> 71 토레스 -> 86 게레로
중위 순회 : 27 루카스 -> 32 펠라인 -> 44 김철수 -> 46 이대호 -> 50 이성진 -> 71 토레스 -> 84 로토스 -> 86 게레로
후위 순회 : 32 펠라인 -> 27 루카스 -> 44 김철수 -> 46 이대호 -> 71 토레스 -> 86 게레로 -> 84 로토스 -> 50 이성진
삭제한 노드 : 32, 펠라인
삭제한 노드 : 27, 루카스
삭제한 노드 : 44, 김철수
삭제한 노드 : 46, 이대호
삭제한 노드 : 71, 토레스
삭제한 노드 : 86, 게레로
삭제한 노드 : 84, 로토스
삭제한 노드 : 50, 이성진
C:\Users\wdnk46\OneDrive\바탕
0 코드로 인해 종료되었습니다
이 창을 닫으려면 아무 키나
        
```

### - 데이터 추가와 중복키가 생겼을 때, 4글자 이름이 입력되었을 때 확인

```

번호가 중복됩니다. (번호 : 46, 이름 : 김동민) 삽입 불가
전위 순회 : 50 이성진 -> 46 이대호 -> 44 김철수 -> 27 루카스 -> 11 가나다 -> 32 펠라인 -> 45 라마바 -> 86 게레로 -> 84 로토스 -> 85 로토스짱 -> 100 백점
중위 순회 : 11 가나다 -> 27 루카스 -> 32 펠라인 -> 44 김철수 -> 45 라마바 -> 46 이대호 -> 50 이성진 -> 71 토레스 -> 84 로토스 -> 85 로토스짱 -> 86 게레로 -> 100 백점
후위 순회 : 11 가나다 -> 32 펠라인 -> 27 루카스 -> 45 라마바 -> 44 김철수 -> 46 이대호 -> 71 토레스 -> 85 로토스짱 -> 84 로토스 -> 100 백점 -> 86 게레로 -> 50 이성진
삭제한 노드 : 11, 가나다
삭제한 노드 : 32, 펠라인
삭제한 노드 : 27, 루카스
삭제한 노드 : 45, 라마바
삭제한 노드 : 44, 김철수
삭제한 노드 : 46, 이대호
삭제한 노드 : 71, 토레스
삭제한 노드 : 85, 로토스짱
삭제한 노드 : 84, 로토스
삭제한 노드 : 100, 백점
삭제한 노드 : 86, 게레로
삭제한 노드 : 50, 이성진
C:\Users\wdnk46\OneDrive\바탕
44 프로세스)이(가) 0 코드로
이 창을 닫으려면 아무 키나
        
```

## 1.5 느낀 점

이번 문제는 학번을 기준으로 이진트리에 저장하고 이를 출력하는 문제였습니다. 링크를 이용한 이진트리문제는 이번에 처음프로그래밍 해보았는데, 이와 관련된 알고리즘을 이해하는데 시간이 꽤 오래 걸렸던 것 같습니다. 특히 마지막 출력에서는 ->표시가 출력이 안 되어야하므로 이 부분을 프로그래밍 하는 부분이 힘들었던 것 같습니다.

처음 프로그램을 짤 때 전위와 중위는 최댓값보다 작을 때 ->표시를 출력하고 후위는 루트노드일 때를 제외하고 ->를 출력하게 했습니다. 하지만 만약 루트노드의 오른쪽 노드가 존재하지 않을 때, 전위 출력에서 마지막에 ->가 출력되는 문제가 생겼습니다. 이런 문제를 확인하고 코드를 완전히 바꾸어 프로그램을 다시 짰습니다.

이 부분을 해결하기 위해 노드의 개수함수를 이용했습니다. static정적 변수를 출력함수에서 선언하여 노드를 하나 둘 때마다 1씩 증가하게 했습니다. 그리고 변수가 노드의 개수보다 작을 때만 ->가 출력되게 프로그래밍 하였습니다. 그래서 제일 마지막에 출력되는 노드는 ->가 출력되지 않을 수 있었습니다.

그리고 이진트리 전체를 삭제하여 메모리의 낭비를 막았습니다. 트리를 어떻게 삭제할까 고민하던 중 후위출력 알고리즘과 비슷하게 프로그래밍 하면 될 것 같아서 코딩해보았습니다. 만약 중간 노드부터 삭제하게 되면 그 밑의 노드는 접근할 수 없기 때문에 가장 밑의 단말 노드부터 삭제했고 이를 순환호출로 프로그래밍 했습니다.

이번 기회로 이진트리의 삽입과 출력에 대해 이해할 수 있는 기회가 되었습니다. 처음에는 마냥 어렵게만 느껴졌었는데 막상 프로그래밍해보니 저도 할 수 있구나 하는 생각이 들었습니다. 아직 많이 부족하지만 앞으로 프로그래밍 공부를 더 열심히 하여 다른 프로그램도 짜보고 싶다는 생각을 했습니다.

## 2. 이진 트리 2

### 완전 이진트리의 여부를 판별하는 프로그램

- data.txt에서 정수로 이루어진 정보를 읽어와 이진트리를 생성하고 생성된 트리가 완전 이진 트리인지 검증하라

#### 1.1 문제 분석

조건 1 파일 data.txt에 저장되어 있는 정보를 사용할 것

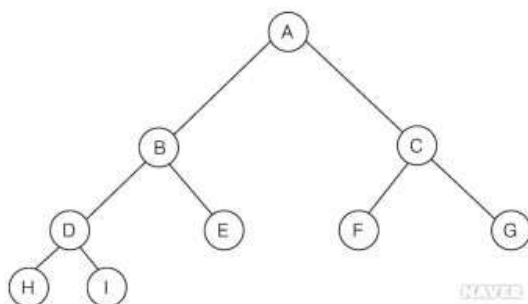
조건 2 insert\_node를 이용하여 트리 생성

조건 3 완전 이진트리인지 아닌지 판별하여 이를 출력

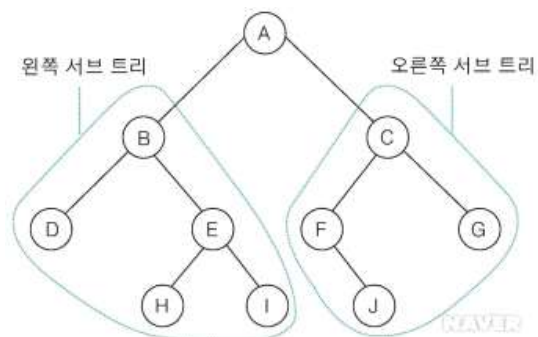
- 이 문제는 data로부터 입력받은 이진트리가 완전 이진 트리인지 판별하는 프로그램이다. 입력된 순서대로 트리에 삽입하는데, 만약 입력받은 숫자가 노드보다 작다면 왼쪽에 저장되고 크다면 오른쪽에 저장될 것이다. 출력은 전위 순회의 순서로 출력되고 만약 완전이진트리가 아니라면 루트노드를 출력한 후 완전이진트리가 아니라는 메시지를 출력하고 종료한다. 완전이진트리의 판별은 노드의 개수로 판단하고, 순환을 이용한다.

**완전 이진 트리** : 완전 이진 트리는 높이가  $k$ 일 때 레벨 1부터  $k-1$ 까지는 노드가 모두 채워져 있고 마지막 레벨  $k$ 에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진트리이다. 마지막 레벨에서는 노드가 꽉차있지 않아도 되지만 중간에 빈곳이 있어서는 안된다.

완전 이진트리인 경우



완전 이진트리가 아닌 경우



## 2.2 소스 코드

```

/*
 * 파일 : 20104012
 * 학과 : 컴퓨터소프트웨어공학부
 * 이름 : 조준환
 * 주일 명 : 이진 탐색과 주일 중 가장 빠르고
 *          빠진 이진 탐색까지 진행하는 프로그램
 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

typedef int Element;

typedef struct Treenode {
    Element data;
    struct Treenode *left, *right;
} Treenode;

int FILLNODE(FillNode, Treenode**);
Treenode* Insert_node(Treenode*, Element);
Treenode* Del_node(Treenode*);
void Preorder(Treenode*, int);
int Del_node_Count(Treenode*);
int Check_Tree(Treenode*, int, int);
void Delete_Tree(Treenode*);

int main() {
    FILE *f;
    int n = 0;
    Treenode* root = NULL;

    if (FILLNODE(f, &root)) return 0;

    printf("Preorder\n");
    if (Check_Tree(root, n, Del_node_Count(root))) {
        Preorder(root, FALSE);
        printf("이진 탐색이 성공적이었습니다.\n");
    }
    else {
        Preorder(root, TRUE);
        printf("이진 탐색이 실패했습니다.\n");
    }
    Delete_Tree(root);
    return 0;
}

int FILLNODE(FillNode, Treenode** root) {
    int n;
    f = fopen("data.txt", "r");
    if (!f) {
        printf("file not opened\n");
        return FALSE;
    }
    while (!feof(f)) {
        fscanf(f, "%d", &n);
        *root = Insert_node(*root, n);
    }
    return TRUE;
}

Treenode* Insert_node(Treenode* node, Element key) {
    if (node == NULL) return new_node(key);

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

    return node;
}

Treenode* Del_node(Treenode* node) {
    Treenode* temp = (Treenode*)malloc(sizeof(Treenode));
    printf("Inorder = %d\n", node->data);
    temp->data = node->data;
    temp->left = temp->right = NULL;
    return temp;
}

int Del_node_Count(Treenode* node) {
    int count = 0;
    if (node == NULL)
        count = 0;
    else
        count = 1 + Del_node_Count(node->left) + Del_node_Count(node->right);

    return count;
}

void Preorder(Treenode* tree, int check) {
    if (tree == NULL) {
        printf("%d", tree->data);
        if (!check) return;
        Preorder(tree->left, check);
        Preorder(tree->right, check);
    }
}

int Check_Tree(Treenode* node, int index, int count) {
    if (node == NULL) return TRUE;
    printf("node %d index %d\n", node->data, index);
    if (index == count) return FALSE;

    if (Check_Tree(node->left, index + 1, count) && Check_Tree(node->right, index + 2, count)) { return TRUE; }
    else return FALSE;
}

void Delete_Tree(Treenode* node) {
    if (node == NULL) {
        Delete_Tree(node->left);
        Delete_Tree(node->right);
        printf("삭제된 노드 : %d\n", node->data);
        free(node);
    }
}

```



## 2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 이진트리에 파일을 저장하고
            완전 이진트리인지 판별하는 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

typedef int Element;

typedef struct TreeNode {
    Element data;
    struct TreeNode *left, *right;
}TreeNode;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다. 참을 나타내는 TRUE와 거짓을 나타내는 FALSE를 define한다.

2. typedef를 이용하여 노드에 들어갈 요소의 자료형을 생성한다. data파일 에 정수 값만 존재하기 때문에 int자료형을 Element로 선언하여 다른 int와 구분할 수 있게 프로그래밍 하였다.

3. 각 항을 하나의 노드로 표현한다. 노드에는 Element 자료형 변수 하나와 왼쪽 노드와 오른쪽 노드를 가리키는 left, right링크가 있다.

```
int FILEOPEN(FILE**, TreeNode**);
TreeNode* Insert_node(TreeNode*, Element);
TreeNode* new_node(Element);
void Preorder(TreeNode*);
int Get_Node_Count(TreeNode*);
int Check_Tree(TreeNode*, int, int);
void Delete_Tree(TreeNode*);
```

3. 필요한 함수를 선언한다.

- FILEOPEN함수는 파일을 열고, 파일이 존재한다면 데이터를 읽어온다.
- Insert\_node함수는 이진탐색트리에 삽입을 하는 함수이다.
- new\_node함수는 동적으로 노드를 할당하여 새로운 노드를 반환하는 함수이다
- Preorder함수는 트리를 전위 순회하여 출력하는 함수이다.

- Get\_Node\_Count함수는 노드의 개수를 구하여 반환하는 함수이다.
- Check\_Tree함수는 트리가 완전이진트리인지 아닌지 판별하는 함수이다.
- Delete\_Tree함수는 트리 안에 있는 노드 전체를 삭제하는 함수이다.

```
int main() {
    FILE *fp;
    int n = 1;
    TreeNode*root= NULL;

    if (!FILEOPEN(&fp, &root))return 0;

    printf("Preorder>>> ");
    if (!Check_Tree(root, n, Get_Node_Count(root))) {
        printf("\n완전 이진탐색트리가 아닙니다.\n");
    }
    else {
        Preorder(root);
        printf("\n완전 이진 탐색트리 입니다.\n");
    }
    Delete_Tree(root);
    fclose(fp);
    return 0;
}
```

#### 4. 필요한 변수들을 선언한다.

- fp는 파일포인터, root는 이진트리를 나타내는 변수이다.
- n은 완전이진트리를 판별할 때 사용되는 변수이다.

#### 5. FILEOPEN함수를 호출하여 파일의 존재여부를 체크한다.

파일이 존재하지 않는다면 함수는 FALSE가 리턴되므로 바로 프로그램을 종료하지만 파일이 존재하면 이진트리에 값을 삽입 후 TRUE가 리턴된다. 값이 유지가 되어야 하기 때문에 fp와 root변수의 주소 값을 전달한다.

#### 6. Check\_Tree함수를 이용하여 이진트리가 완전이진트리인지 판별한다.

매개변수로는 트리와 n값(1), 노드의 개수(Get\_Node\_Count)값을 전달한다. 만약 완전이진트리가 아니라면 전위 출력 함수를 호출하지 않고 맞다면 전위함수 호출 후 그에 맞는 메시지를 출력한다.

#### 7. 출력이 종료되면 Delete\_Tree함수로 이진트리에 있는 노드를 전부 삭제한다. fclose함수로 파일을 닫은 후 프로그램을 종료한다.



```

int FILEOPEN(FILE**fp, TreeNode**root) {
    int n;
    *fp = fopen("data.txt", "r");
    if (!*fp) {
        printf("file not open\n");
        return FALSE;
    }
    while (!feof(*fp)) {
        fscanf(*fp, "%d", &n);
        *root = Insert_node(*root, n);
    }
    return TRUE;
}

```

8. FILEOPEN함수는 파일을 열고 파일이 존재한다면 값을 읽어오는 함수이다. 매개변수로는 파일포인터의 주소값과 트리의 주소값을 전달받는다.

9. 파일 오픈 후 만약 파일이 존재하지 않는다면 file not open이라는 메시지를 출력하고 FALSE를 리턴한다.

파일의 존재한다면 파일로부터 data를 읽어온다. feof함수를 이용하여 끝까지 data를 읽고 이를 이진트리에 저장한다.

10. insert\_node함수는 새로운 노드 또는 변경된 루트 포인터를 반환하므로 root변수에 이 값을 받아주도록 한다. 입력이 끝나면 TRUE를 리턴한다.

```

TreeNode* Insert_node(TreeNode* node, Element key) {
    if (node == NULL) return new_node(key);

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

    return node;
}

TreeNode* new_node(Element item) {
    TreeNode* temp = (TreeNode*)malloc(sizeof(TreeNode));
    printf("inserted -> %d\n", item);
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

```

11. Insert\_node함수는 이진트리에 노드를 삽입하는 함수이다. 매개 변수로는 트리와 삽입할 key값을 전달받는다.

13. 만약 트리가 공백이면 새로운 노드를 반환한다. 트리가 공백이 아니라면 순환적으로 트리를 내려가는데, 만약 key가 노드 데이터보다 작다면 왼쪽으로 내려가고 크다면 오른쪽으로 내려간다.

새로운 노드나 변경된 루트포인터 리턴값을 left또는 right에 입력받는다.

14. new\_node함수는 동적으로 메모리를 할당하여 새로운 노드를 생성 후 반환하는 함수이다. 임시 노드 temp를 동적할당 후 값을 넣고 left와 right 값을 NULL로 설정한 다음 이를 리턴한다.

```
int Get_Node_Count(TreeNode*node) {
    int count = 0;
    if (node != NULL)
        count = 1 + Get_Node_Count(node->left) + Get_Node_Count(node->right);

    return count;
}

void Preorder(TreeNode* tree) {
    if (tree != NULL) {
        printf("%d ", tree->data);
        Preorder(tree->left);
        Preorder(tree->right);
    }
}
```

15. Get\_Node\_Count함수는 이진트리의 노드 개수를 리턴하는 함수이다. 노드의 개수를 세기 위해서는 트리안의 노드들을 전체적으로 순회한다. 각각의 서브트리에 대하여 순환 호출한 다음, 반환되는 값에 1을 더하여 반환한다.

16. Preorder함수는 트리를 전위 순회하여 출력하는 함수이다. 먼저 부모노드가 출력되고 왼쪽 노드, 오른쪽 노드 순으로 출력된다.

```
int Check_Tree(TreeNode*node, int index, int count) {
    if (node == NULL) return TRUE;
    // printf("node->%d index->%d\n", node->data, index);
    if (index > count) return FALSE;

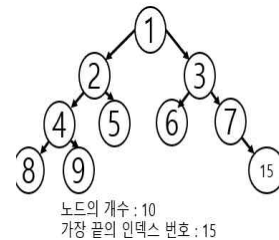
    if (Check_Tree(node->left, 2 * index, count) && Check_Tree(node->right, 2 * index + 1, count))
        return TRUE;
    else return FALSE;
}

void Delete_Tree(TreeNode*node) {
    if (node != NULL) {
        Delete_Tree(node->left);
        Delete_Tree(node->right);
        printf("삭제 한 노드 : %d\n", node->data);
        free(node);
    }
}
```

18. Check\_Tree함수는 트리가 완전이진트리인지 판별하는 함수이다.  
매개 변수는 트리와, 인덱스 번호, 노드의 개수를 전달받는다.  
배열 트리를 구현하는 방법을 이용하여 프로그래밍 하였다.

19. 현재 노드가 NULL이라면 TRUE를 리턴한다.  
완전이진트리 마지막 k레벨에서 인덱스 번호가 노드의 개수보다 많아진다면 완전 이진트리가 아니라는 말이므로 FALSE를 리턴한다.

20.인덱스 번호는 노드의 개수보다 같거나 작아야 한다.  
오른쪽 사진과 같이 노드는 10개지만 가장 끝의 인덱스 번호가 15이기 때문에 이 트리는 완전 이진트리가 될 수 없다.(트리 안의 숫자는 인덱스 번호이다.)



21. 이렇게 배열의 알고리즘을 이용하여 수행하기 위해 메인함수에서 n 값을 1로 설정했다. 왼쪽 노드에는 인덱스 번호에 \*2를 하고 오른쪽 노드는 \*2+1하여 순환을 이용하여 호출하는데 아래 노드가 모두 TRUE를 리턴하면 부모노드도 TRUE를 리턴한다.

22. Delete\_Tree함수는 이진트리 전체를 삭제하는 함수이다.  
부모 노드를 먼저 삭제하게 되면 그 밑의 노드들은 접근할 수 없기 때문에 가장 아래에 있는 단말 노드부터 삭제한다. 따라서 왼쪽 자식노드, 오른쪽 자식노드를 삭제하고 부모 노드를 삭제한다.

23. 어떤 노드가 삭제되는지 출력하고 free로 노드를 삭제한다.  
결과적으로 루트노드가 마지막으로 삭제될 것이고 모든 노드가 메모리 해제 될 것이다.

## 2.4 실행 창

- 문제의 데이터 값으로 출력

문제 data1	문제 data2
<pre> inserted -&gt; 8 inserted -&gt; 3 inserted -&gt; 10 inserted -&gt; 14 inserted -&gt; 2 inserted -&gt; 5 inserted -&gt; 9 inserted -&gt; 11 inserted -&gt; 16 inserted -&gt; 4 inserted -&gt; 6 Preorder&gt;&gt;&gt; 완전 이진탐색트리가 아닙니다. </pre> <div> data - Windows 메모장 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H) 8 3 10 14 2 5 9 11 16 4 6 </div>	<pre> inserted -&gt; 8 inserted -&gt; 3 inserted -&gt; 10 inserted -&gt; 14 inserted -&gt; 2 inserted -&gt; 5 inserted -&gt; 9 Preorder&gt;&gt;&gt; 8 3 2 5 10 9 14 완전 이진 탐색트리 입니다. </pre> <div> data - Windows 메모장 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H) 8 3 10 14 2 5 9 </div>

- 삭제한 노드 출력과 다른 데이터로 완전 이진트리 판별

7노드의 오른쪽이 비었을 때 (완전이진트리x)	8을 추가, 오른쪽 채웠을 때 (완전이진트리o)
<pre> inserted -&gt; 10 inserted -&gt; 5 inserted -&gt; 20 inserted -&gt; 2 inserted -&gt; 7 inserted -&gt; 17 inserted -&gt; 25 inserted -&gt; 1 inserted -&gt; 6 inserted -&gt; 3 inserted -&gt; 15 Preorder&gt;&gt;&gt; 완전 이진탐색트리가 아닙니다. 삭제한 노드 : 1 삭제한 노드 : 3 삭제한 노드 : 2 삭제한 노드 : 6 삭제한 노드 : 7 삭제한 노드 : 5 삭제한 노드 : 15 삭제한 노드 : 17 삭제한 노드 : 25 삭제한 노드 : 20 삭제한 노드 : 10 </pre> <div> data - Windows 메모장 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H) 10 5 20 2 7 17 25 1 6 3 15 </div>	<pre> inserted -&gt; 10 inserted -&gt; 5 inserted -&gt; 20 inserted -&gt; 2 inserted -&gt; 7 inserted -&gt; 17 inserted -&gt; 25 inserted -&gt; 1 inserted -&gt; 6 inserted -&gt; 3 inserted -&gt; 15 inserted -&gt; 8 Preorder&gt;&gt;&gt; 10 5 2 1 3 7 6 8 20 17 15 25 완전 이진 탐색트리 입니다. 삭제한 노드 : 1 삭제한 노드 : 3 삭제한 노드 : 2 삭제한 노드 : 6 삭제한 노드 : 7 삭제한 노드 : 5 삭제한 노드 : 15 삭제한 노드 : 17 삭제한 노드 : 25 삭제한 노드 : 20 삭제한 노드 : 10 </pre> <div> data - Windows 메모장 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H) 10 5 20 2 7 17 25 1 6 3 15 8 </div>

## 2.5 느낀 점

이번 문제는 정수가 저장된 데이터를 읽어 와서 이진트리에 저장하고 이 트리가 완전이진트리인지 아닌지 판별하는 프로그램이었습니다. 삽입이나 출력같이 다른 함수들은 1번 문제와 비슷한 함수들이 많았지만, 완전이진트리 판별 함수는 처음 접하는 알고리즘이었기 때문에 이를 프로그래밍 하는 것이 매우 힘들었던 것 같습니다.

완전이진트리 판별함수를 만들기 위해서 먼저 노드의 개수를 생각했습니다. 그리고 배열로 구현한 이진트리를 수행할 때했던 인덱스 알고리즘을 이용하여 프로그래밍 해보았습니다. 배열 트리를 구현할 때 현재 노드에  $*2$ 를 하면 왼쪽 노드,  $*2+1$ 을 하면 오른쪽 노드의 인덱스번호를 구할 수 있었습니다. 인덱스 번호로는 노드의 위치뿐만 아니라 개수까지 알 수 있었기 때문에 이 점을 이용하여 프로그래밍 해보았습니다.

그래서 배열 트리를 이용할 때처럼 처음 루트 값의 인덱스를 1로 설정하고 왼쪽은  $*2$ , 오른쪽은  $*2+1$ 로 이동했고 만약 이 인덱스 값이 노드의 개수보다 크다면 완전이진트리가 아닌 것으로 코딩했습니다. 왼쪽과 오른쪽의 서브트리가 TRUE를 리턴하면 부모노드도 TRUE를 리턴하고 만약 하나라도 FALSE를 리턴하면 완전이진트리가 아닌 것으로 했습니다. 처음에 이 부분을 구현하는 것이 가장 힘들었는데 순환을 이용하여 프로그래밍하니 생각보다 쉽게 코드를 짤 수 있었던 것 같습니다.

이 프로그램을 처음 짤 때는 전위 출력 함수에서 숫자 하나만 출력하고 종료하는 것이 이해가 되지 않았는데, 이 부분에 대해서 교수님께서 다른 방법이 있다면 그거로 하면 된다고 해주셔서 프로그램을 마칠 수 있었습니다.

이번 기회를 통해 완전이진트리가 맞는지 아닌지 판별하는 함수에 대해서 알 수 있었고 순환 호출에 대해서도 조금이나마 이해할 수 있었던 기회가 되었습니다. 앞으로 트리 공부를 더 열심히 해서 다른 어려운 문제들도 풀 수 있도록 노력하겠습니다.

### 3. 이진 트리를 하며 느낀 점

이번 이진트리 문제 두 가지는 링크를 이용하여 구현한 이진트리로 처음 접하는 알고리즘도 몇 가지 있었습니다. 처음 코딩을 했을 때는 어렵다고 생각했던 부분도 있었는데 프로그래밍을 해보니 쉽게 할 수 있었던 부분도 많았습니다. 이번 기회를 계기로 트리 구조에 대해 더 배우고 싶은 부분이 많아졌고 다른 문제들도 풀어보고 싶다는 생각을 했습니다.

이번에 레포트를 쓰며 프로그래밍 할 때 미처 생각하지 못한 부분들을 알 수 있었고 어떤 부분은 더 좋게 바꿀 수 있을 것 같아 바꾼 부분도 있었습니다. 이를 계기로 코드를 설명한다는 것이 얼마나 중요한 일인지 알 수 있었습니다. 코드를 설명하며 원래 코드보다 더 좋은 알고리즘을 찾을 수 있었고, 나 하나만 이해하는 것이 아니라 이 코드를 보는 다른 사람을 이해시키기 위해서는 내가 정확히 알고 있어야 한다는 것을 깨달을 수 있었습니다.

특히 완전이진트리 판별함수에서 꽤 많은 시간이 걸렸는데 이 부분이 성공적으로 되니 성취감을 느낄 수 있었습니다. 앞으로는 책에 나와 있지 않은 여러 알고리즘들도 만들어 보고 싶다는 생각을 했습니다. 이를 실천하기 위해 트리 공부를 더 많이 해야겠다고 다짐했습니다.

트리 알고리즘은 자료구조 중에서도 가장 중요한 알고리즘이라고 배웠기 때문에 놓치는 점 하나 없이 수업뿐만 아니라 과제 또한 열심히 해야겠다고 다짐했습니다. 앞으로 더 어려운 알고리즘을 배우게 될 것 같은데 꾸준히 공부하여 좋은 점수 받을 수 있도록 노력하겠습니다.

이상 “이진트리” 레포트를 마치겠습니다.

- 감사합니다. -