



순천향대학교
SOON CHUN HYANG
UNIVERSITY

자료구조 실습 HW 4

자료구조 2

담당교수	홍민 교수님
학과	컴퓨터소프트웨어공학과
학번	20204005
이름	김필중
제출일	2021년 09월 28일

| 목 차 |

1. 재귀 순환을 이용한 이진 트리

- 1.1 문제 분석
- 1.2 소스 코드
- 1.3 소스 코드 분석
- 1.4 실행 화면
- 1.5 느낀점

2. 완전 이진트리 판별

- 2.1 문제 분석
- 2.2 소스 코드
- 2.3 소스 코드 분석
- 2.4 실행 화면
- 2.5 느낀점


3. 반복을 이용한 이진 트리

- 3.1 문제 분석
- 3.2 소스 코드
- 3.3 소스 코드 분석
- 3.4 실행 화면
- 3.5 느낀점

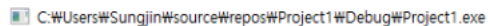
1.1 문제 분석

이진 트리 1

- 파일 data.txt에 아래와 같은 학번과 이름들이 저장되어 있다.
 - data.txt 파일에서 데이터를 입력
 - 학번을 이용하여 insert_node함수로 데이터를 이진 트리에 저장
 - 저장된 정보를 전위, 중위, 후위순으로 출력



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



```
C:\Users\Sungjin\source\repos\Project1\Debug\Project1.exe
```

```
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
```

이번 1번 과제는 이진 트리에서 배열이 아닌 연결리스트 구조체를 이용하여 트리를 만들어야 한다.

첫번째로는 학생 구조체를 만들어야 한다. 학생 구조체 안에는 학번을 저장하는 변수, 이름을 저장하는 변수 2개를 만들어 준다.

두번째로는 트리 구조체를 만들어 준다. 구조체에는 트리의 좌 우 링크와 학생 구조체 data를 넣어서 만들어 준다.

다음으로는 노드 삽입 함수를 만들어준다. 노드 삽입은 우선 빈 노드일 경우는 노드를 동적할당해 데이터의 값을 넣어주고 만약 아닐경우는 크기를 비교해 좌 우 링크로 이동해 데이터를 넣어주는 함수를 만들어 준다.

노드 순회 함수를 만들어 준다. 전위, 중위, 후위 순회 함수를 만들어서 데이터를 출력해야 한다. 그러기 위해서는 우선 재귀함수를 통해 만들어 준다. 각 순회 함수에 맞춰서 재귀 함수를 배치하는 것이 중요 포인트다. 여기서 또 추가할 사항은 마지막 데이터를 출력하는 부분일 경우 화살표를 제거해야한다. 그러기에 현

재 데이터의 개수와 마지막 번째의 데이터 개수의 크기를 비교해 데이터를 출력하는 방식을 택할 것이다. 그러기에 이번에는 전역 변수를 하나 사용해야한다.

마지막으로는 노드의 개수를 세는 함수를 만들어 준다. 이 또한 재귀함수를 통해 반복을 통해 데이터의 개수를 세어 반환한다.

1.2 소스 코드

```
1  //-----
2  // 제작 기간 : 21년 09월 22일 ~ 21년 09월 26일
3  // 제작자 : 20204005 김필중
4  // 프로그램명 : 순회를 통한 이진트리 순회
5  //-----
6
7
8  // 필요한 헤더파일을 선언한다
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12
13 // 오류 방지 구문
14 #pragma warning (disable : 4996)
15
16 // 전역 변수 check 를 선언한다
17 int check = 0;
18
19 // student 학생 구조체를 만든다
20 typedef struct student
21 {
22     int id;
23     char name[30];
24 }Student;
25
26 // 트리 구조체를 만든다
27 typedef struct TreeNode
28 {
29     Student key; // 트리의 정보를 가지고 있는 구조체
30     struct TreeNode *left, *right; // 좌 우 트리 연결
31 }TreeNode;
32
33 // 새로운 노드 생성 함수
34 TreeNode * new_node(Student data)
35 {
36     TreeNode *temp = (TreeNode *)malloc(sizeof(TreeNode)); // temp 노드에 treenode 크기만큼 동적할당 해준다
37     temp->key.id = data.id; // 학번 입력
38     strcpy(temp->key.name, data.name); // 문자열은 strcpy 를 통해 입력
39     temp->left = temp->right = NULL; // 노드의 좌 우는 null로 설정해준다
40     return temp; // 반환한다
41 }
42
43
```

```

44 // 노드 삽입 함수
45 TreeNode * insert_node(TreeNode * node, Student data)
46 {
47     if (node == NULL) // 만약 노드가 없을 경우는
48         return new_node(data); // 새로운 노드를 만들어 준다
49
50     if (node->key.id > data.id) // 현재 노드의 학번이 받은 학번의 크기보다 크다면
51         node->left = insert_node(node->left, data); // 왼쪽에 저장해준다
52     else if (node->key.id < data.id) // 반대일 경우는
53         node->right = insert_node(node->right, data); // 오른쪽에 저장한다
54
55     return node; // 노드를 반환한다
56 }
57
58 // 전위 순회 함수
59 void preorder(TreeNode *root, int cnt)
60 {
61     if (root != NULL) // 만약 노드가 널이 아닐 경우는
62     {
63         if (cnt-1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
64         {
65             printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
66             return; // 함수 종료
67         }
68         printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
69         check++; // 전역 변수에 하나를 더해준다.
70         // 전위 순회 함수 재귀 함수를 호출한다
71         preorder(root->left, cnt);
72         preorder(root->right, cnt);
73     }
74 }
75
76 // 중위 순회 함수
77 void inorder(TreeNode *root, int cnt)
78 {
79
80     if (root != NULL) // 만약 노드가 널이 아닐 경우는
81     {
82         // 중위 순회 함수 재귀 함수를 호출한다
83         inorder(root->left, cnt);
84         if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
85         {
86             printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
87             return; // 함수 종료
88         }
89         printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
90         check++; // 전역 변수에 하나를 더해준다.
91         // 중위 순회 함수 재귀 함수를 호출한다
92         inorder(root->right, cnt);
93     }
94 }

```

```

97 // 후위 순회 함수
98 void postorder(TreeNode *root, int cnt)
99 {
100     if (root != NULL) // 만약 노드가 널이 아닐경우는
101     {
102         // 후위 순회 함수 재귀 함수를 호출한다
103         postorder(root->left, cnt);
104         postorder(root->right, cnt);
105         if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
106         {
107             printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
108             return; // 함수 종료
109         }
110         printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
111         check++; // 전역 변수에 하나를 더해준다.
112     }
113 }
114
115 // 노드의 총 개수를 파악하는 함수
116 int get_node_count(TreeNode *node)
117 {
118     int count = 0;
119
120     if (node != NULL)
121     {
122         count = 1 + get_node_count(node->left) + get_node_count(node->right); // 노드 1 + 재귀 함수를 통해 좌 우의 개수를 파악해 count에 저장한다
123     }
124     return count; // 반환한다
125 }
126

```



```

128 int main(void)
129 {
130     TreeNode *root = NULL; // 루트 노드를 생성한다
131     Student s; // 구조체를 생성한다
132     FILE *fp; // 파일 포인터 fp를 선언한다
133
134     char temp[30]; // 이름 임시 저장소
135     int cnt = 0; // 총 노드 저장소
136
137     fp = fopen("data.txt", "r"); // 파일을 오픈한다
138
139     if (fp == NULL) // 오픈이 안될경우 종료한다
140     {
141         printf("파일 오픈 실패");
142         return 0;
143     }
144
145     // 파일 끝까지 반복한다
146     while (!feof(fp))
147     {
148         fscanf(fp, "%d %s\n", &s.id, temp); // 파일에서 읽어 와서 저장을 한다
149         strcpy(s.name, temp); // 문자열은 strcpy 로 저장한다
150         root = insert_node(root, s); // root 노드에 값을 넣는다
151     }
152
153     cnt = get_node_count(root); // 총 노드 개수 함수를 호출한다
154
155     // 전위 함수
156     printf("전위 순회 :");
157     preorder(root, cnt);
158     printf("\n");
159
160     // 중위 함수
161     check = 0; // 전역 변수 check를 초기화 한다
162     printf("중위 순회 :");
163     inorder(root, cnt);
164     printf("\n");
165
166     // 후위 함수
167     check = 0; // 전역 변수 check를 초기화 한다
168     printf("후위 순회 :");
169     postorder(root, cnt);
170     printf("\n");
171
172
173     fclose(fp); // 파일을 닫는다
174     free(root); // 동적할당 해제한다
175     return 0; // 종료한다.
176 }

```

1.3 소스 코드 분석

```
8      // 필요한 헤더파일을 선언한다
9      #include <stdio.h>
10     #include <stdlib.h>
11     #include <string.h>
12
13     // 오류 방지 구문
14     #pragma warning (disable : 4996)
15
```

1. 필요한 헤더 파일을 생성한다
string.h 역할은 strcpy 를 사용하기 위함이다.
2. 오류 방지 구문을 선언한다.

```
// 전역 변수 check 를 선언한다
int check = 0;
```

3. 마지막 데이터를 출력할 때 확인하기 위한 전역변수를 선언한다.

```
// student 학생 구조체를 만든다
typedef struct student
{
    int id;
    char name[30];
}Student;
```

4. 학생 구조체를 만든다.
5. 학생 학번 변수, 이름 변수를 선언한다.

```
// 트리 구조체를 만든다
typedef struct TreeNode
{
    Student key; // 트리의 정보를 가지고 있는 구조체
    struct TreeNode *left, *right; // 좌 우 트리 연결
}TreeNode;
```

6. 트리 구조체를 만든다.

7. 트리의 정보를 가지고 있는 구조체 key를 선언한다 여기에 학생 정보를 넣어 준다.

8. 자기 참조를 통해 좌 우 링크를 만들어 준다. 이걸 통해 트리 밑으로 쪽 뻗어나갈 수 있다.

```
// 새로운 노드 생성 함수
TreeNode * new_node(Student data)
{
    TreeNode *temp = (TreeNode *)malloc(sizeof(TreeNode)); // temp 노드에 treeNode 크기만큼 동적할당 해준다
    temp->key.id = data.id; // 학번 입력
    strcpy(temp->key.name, data.name); // 문자열은 strcpy 를 통해 입력
    temp->left = temp->right = NULL; // 노드의 좌 우는 null로 설정해준다
    return temp; // 반환한다
}
```

9. 새로운 노드 생성 함수를 만든다. 이 함수는 현재 노드가 null 일 경우 동적할당을 해준다.

10. 동적 할당한 노드에 학번을 넣는다.

11. 문자열은 strcpy를 통해 데이터를 삽입해준다.

12. 트리 노드여서 좌 우 링크가 생겼기 때문에 null 로 설정해준다.

```
// 노드 삽입 함수
TreeNode * insert_node(TreeNode * node, Student data)
{
    if (node == NULL) // 만약 노드가 없을경우는
        return new_node(data); // 새로운 노드를 만들어 준다

    if (node->key.id > data.id) // 현재 노드의 학번이 받은 학번의 크기보다 크다면
        node->left = insert_node(node->left, data); // 왼쪽에 저장해준다
    else if (node->key.id < data.id) // 반대일 경우는
        node->right = insert_node(node->right, data); // 오른쪽에 저장한다

    return node; // 노드를 반환한다
}
```

13. 노드 삽입함수를 만든다. 만약 현재 노드가 없을 경우는 새로운 노드를 만들어준다.

14. 현재 노드의 학번이 받은 학번이 크기보다 크다면 왼쪽 아닐경우는 오른쪽에

저장한다.

```
// 전위 순회 함수
void preorder(TreeNode *root, int cnt)
{
    if (root != NULL) // 만약 노드가 널이 아닐경우는
    {
        if (cnt-1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
        {
            printf("%d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
            return; // 함수 종료
        }
        printf("%d %s ->", root->key.id, root->key.name); // 값을 출력한다
        check++; // 전역 변수에 하나를 더해준다.
        // 전위 순회 함수 재귀 함수를 호출한다
        preorder(root->left, cnt);
        preorder(root->right, cnt);
    }
}
```

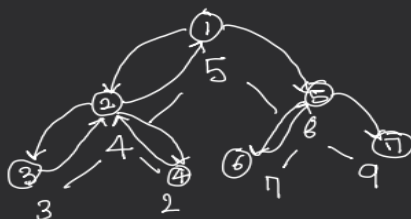
15. 전위 순회 함수는 부모 노드부터 왼쪽 순서 다시 리턴 오른쪽으로 가는 순회이다.

16. 노드가 널이 아니고 만약 현재 노드의 개수 -1이 전체 개수랑 같을경우는 마지막 데이터 값을 화살표 없이 출력하고 종료한다.

17. 아닐 경우는 값을 출력하고 전역변수에 하나를 더해준다

18. 전위 순회 함수 재귀 함수를 루트 노드의 좌 우 값을 넣어 호출해준다.

전위 순회



만약 $(\text{총 개수} - 1) == \text{현재 위치까지의 개수}$

즉 다음이 마지막이므로

→ 없이 출력

총 노수 = 3

2이 도착한 후 확인

$$(3-1) == 2$$

이므로 출력은

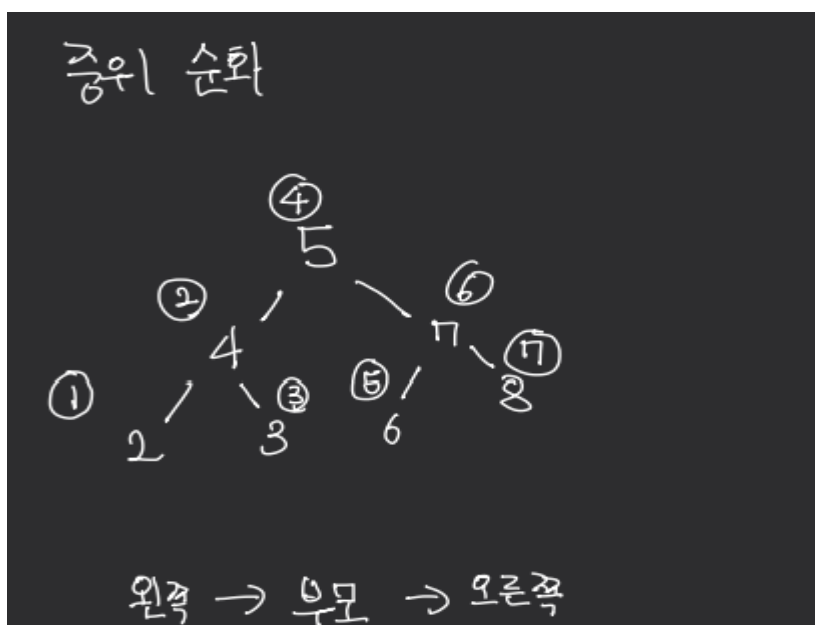
1 → 2 → 3

```

// 중위 순회 함수
void inorder(TreeNode *root, int cnt)
{
    if (root != NULL) // 만약 노드가 널이 아닐경우는
    {
        // 중위 순회 함수 재귀 함수를 호출한다
        inorder(root->left, cnt);
        if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
        {
            printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
            return; // 함수 종료
        }
        printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
        check++; // 전역 변수에 하나를 더해준다.
        // 중위 순회 함수 재귀 함수를 호출한다
        inorder(root->right, cnt);
    }
}

```

19. 중위 순회 함수는 왼쪽 노드부터 부모노드 오른쪽 노드로 이동하면서 프린트 한다.
20. 중위 순회 함수 재귀 함수를 루트 노드의 좌 값을 넣어 호출해준다.
21. 노드가 널이 아니고 만약 현재 노드의 개수 -1이 전체 개수랑 같을경우는 마지막 데이터 값을 화살표 없이 출력하고 종료한다.
22. 아닐 경우는 값을 출력하고 전역변수에 하나를 더해준다
23. 중위 순회 함수 재귀 함수를 루트 노드의 우 값을 넣어 호출해준다.

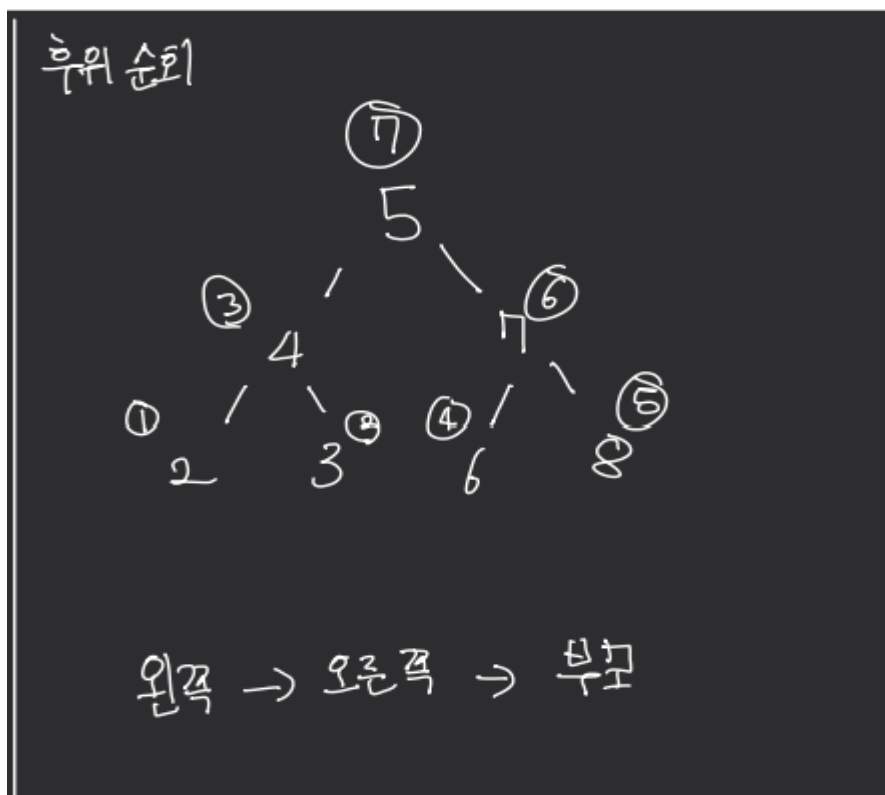


```

// 후위 순회 함수
void postorder(TreeNode *root, int cnt)
{
    if (root != NULL) // 만약 노드가 널이 아닐경우는
    {
        // 후위 순회 함수 재귀 함수를 호출한다
        postorder(root->left, cnt);
        postorder(root->right, cnt);
        if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
        {
            printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
            return; // 함수 종료
        }
        printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
        check++; // 전역 변수에 하나를 더해준다.
    }
}

```

24. 후위 순회 함수는 왼쪽노드 오른쪽 노드 부모 노드순으로 방문한다
25. 후위 순회 함수 재귀 함수를 루트 노드의 좌 우 값을 넣어 호출해준다.
26. 아닐 경우는 값을 출력하고 전역변수에 하나를 더해준다
27. 노드가 널이 아니고 만약 현재 노드의 개수 -1이 전체 개수랑 같을경우는 마지막 데이터 값을 화살표 없이 출력하고 종료한다.



```
// 노드의 총 개수를 파악하는 함수
int get_node_count(TreeNode *node)
{
    int count = 0;

    if (node != NULL)
    {
        count = 1 + get_node_count(node->left) + get_node_count(node->right); // 노드 1 + 재귀 함수를 통해 좌 우의 개수를 파악해 count에 저장한다
    }
    return count; // 반환한다
}
```

28. 노드의 총개수를 파악하기 위해서는 count 변수에 각각에 서브 트리에 대해 순환 호출하고 반환되는 값에 1을 더한다.

```
int main(void)
{
    TreeNode *root = NULL; // 루트 노드를 생성한다
    Student s; // 구조체를 생성한다
    FILE *fp; // 파일 포인터 fp를 선언한다
}
```

29. 루트 노드를 생성한다

30. 학생 구조체를 생성하고 파일포인터를 선언한다.

```
fp = fopen("data.txt", "r"); // 파일을 오픈한다

if (fp == NULL) // 오픈이 안될경우 종료한다
{
    printf("파일 오픈 실패");
    return 0;
}
```

31. 파일을 오픈하고 실패할 경우 종료한다.

```
// 파일 끝까지 반복한다
while (!feof(fp))
{
    fscanf(fp, "%d %s\n", &s.id, temp); // 파일에서 읽어 와서 저장을 한다
    strcpy(s.name, temp); // 문자열은 strcpy 로 저장한다
    root = insert_node(root, s); // root 노드에 값을 넣는다
}

cnt = get_node_count(root); // 총 노드 개수 함수를 호출한다
```

32. 파일 끝까지 반복한다

33. 파일을 읽어와서 저장을 한 후 문자열은 strcpy를 통해 값을 넣어준다

34. root 노드에 값을 넣는 insert 함수를 이용한다

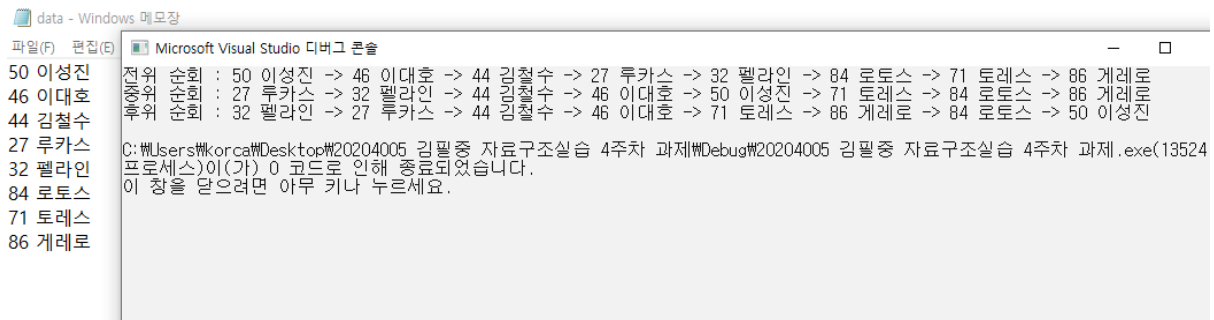
35. 총 노드 개수 함수를 통해 cnt값을 저장한다

```
fclose(fp); // 파일을 닫는다  
free(root); // 동적할당 해제한다  
return 0; // 종료한다.
```

36. 파일을 닫고 동적할당을 해제한다

37. 종료한다,

1.4 실행 화면



```
data - Windows 메모장
Microsoft Visual Studio 디버그 콘솔
파일(F) 편집(E)
50 이성진 전위 순회 : 50 이성진 -> 46 이대호 -> 44 김철수 -> 27 루카스 -> 32 펠라인 -> 84 로토스 -> 71 토레스 -> 86 게레로
46 이대호 중위 순회 : 27 루카스 -> 32 펠라인 -> 44 김철수 -> 46 이대호 -> 50 이성진 -> 71 토레스 -> 84 로토스 -> 86 게레로
44 김철수 후위 순회 : 32 펠라인 -> 27 루카스 -> 44 김철수 -> 46 이대호 -> 71 토레스 -> 86 게레로 -> 84 로토스 -> 50 이성진
27 루카스 C:\Users\korca\Desktop\20204005 김필중 자료구조실습 4주차 과제\Debug\20204005 김필중 자료구조실습 4주차 과제.exe(13524
32 펠라인 프로세스)이(가) 0 코드로 인해 종료되었습니다.
84 로토스 이 창을 닫으려면 아무 키나 누르세요.
71 토레스
86 게레로
```

1.5 느낀점

전에 진행했던 과제와는 다르게 이번에는 이진 트리를 배열이 아닌 연결리스트를 이용해 만들어야 했다. 그래서 그런지 이번에는 그림의 도움이 매우 필요했었다. 우선 순회의 진행 순서부터 노드가 들어가는 것까지 책에 있는 글로만으로는 이해하기 힘들어서 그림을 통해 최대한 여러가지 경우를 상상하면서 코드를 작성했다. 중간중간 이해하기 힘든 부분들을 그림을 통해 최대한 이해하려고 노력했고 많은 도움을 받았다. 이진 트리를 배열이 아닌 연결리스트를 이용해 만들면 장점은 역시 삽입 혹은 삭제가 배열에 비해 단순하다는 것이다. 배열은 많은 수들이 이동해야 하지만 연결리스트는 간단하게 링크만 옮기면 되는 것이기에 더 쉽게 느껴졌다. 물론 이 문제에서는 나오지는 않았지만 만약 시험이나 내가 프로그램 프 제작할 때 트리가 사용하면 아마 삭제 삽입이 자주 사용되는 연결리스트로 제작해야 한다 생각한다. 또한 현재는 재귀 순환으로 만들었지만 반복으로도 만들 수 있다.

2.1 문제 분석

이진 트리 2

- data.txt에서 정수로 이루어진 정보를 읽어와 이진 트리를 생성하고 생성된 트리가 완전 이진 트리인지 아닌지 검증하시오.
- 파일 data.txt에 저장되어 있는 정보를 사용할 것
- insert_node를 이용하여 트리 생성

data1.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움말
8 3 10 14 2 5 9 11 16 4 6

data2.txt

파일(F) 편집(E) 서식(O) 보기(V) 도
8 3 10 14 2 5 9

C:\WINDOWS\system32\cmd.exe

```
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
Inserted 11
Inserted 16
Inserted 4
Inserted 6
Preorder >> 8
완전 이진 탐색트리가 아닙니다. 계속해
```

C:\WINDOWS\system32\cmd.exe

```
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
Preorder >> 8 3 2 5 10 9 14
완전 이진 탐색트리입니다.
계속하려면 아무 키나 누르십시오 . . .
```

이번 문제는 이진트리에서 완전 이진트리인지 아닌지를 확인해야 하는 코드를 만들어야 한다. 완전 이진트리는 높이가 k 일 때 $k-1$ 까지 모두 채워져있고 마지막 k 레벨일때는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진트리이다. 마지막 레벨에서는 노드가 다 안채워져있어도 되지만 중간에 빈 곳이 있으면 안된다

우선 트리 구조체를 만들어야 한다. 이번 트리 구조체는 자기 참조를 통해 오른쪽 왼쪽 링크를 만들고 int형 변수를 만든다.

다음으로는 노드 삽입 함수를 만들어준다. 노드 삽입은 우선 빈 노드일 경우는 노드를 동적할당해 데이터의 값을 넣어주고 만약 아닐경우는 크기를 비교해 좌우 링크로 이동해 데이터를 넣어주는 함수를 만들어 준다.

노드의 개수를 세는 함수를 만들어 준다. 이 또한 재귀함수를 통해 반복을 통해 데이터의 개수를 세어 반환한다.

마지막으로는 완전 이진트리 인지 확인하는 함수를 만들어 준다. 우선 노드의 총 개수를 세어 준 후 먼저 현재 노드가 null 이면 참으로 리턴 해준다. 아닐 경우는 만약 현재 노드의 위치가 총 노드의 수보다 크거나 같을 경우는 거짓을 반환한다. 좌 우를 비교해서 만약 둘 다 참일 경우는 참인 1을 반환하고 거짓일 경우는 0을 반환한다.

2.2 소스 코드

```
1  //-----
2  // 제작 기간 : 21년 09월 22일 ~ 21년 09월 26일
3  // 제작자 : 20204005 김필중
4  // 프로그래밍 : 완전 이진트리 판별
5  //-----
6
7  // 필요한 헤더파일을 선언한다
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 // 오류 방지 구문
12 #pragma warning(disable : 4996)
13
14 // 트리 구조체를 만든다
15 typedef struct TreeNode
16 {
17     int key; // 트리의 노드 값
18     struct TreeNode *left, *right; // 좌 우 트리 연결
19 }TreeNode;
20
21 // 새로운 노드 생성 함수
22 TreeNode * new_node(int data)
23 {
24     TreeNode *temp = (TreeNode *)malloc(sizeof(TreeNode)); // temp 노드에 treeNode 크기만큼 동적할당 해준다
25     temp->key = data; // 받은 값을 입력한다
26     temp->left = temp->right = NULL; // 노드의 좌 우는 null로 설정해준다
27     return temp;
28 }
29
30 // 노드 삽입 함수
31
32 TreeNode * insert_node(TreeNode * node, int data)
33 {
34     if (node == NULL) // 만약 노드가 없을 경우는
35         return new_node(data); // 새로운 노드를 만들어 준다
36
37     if (node->key > data) // 현재 노드의 크기가 받은 데이터의 크기보다 크다면
38         node->left = insert_node(node->left, data); // 왼쪽에 저장해준다
39     else if (node->key < data) // 반대일 경우는
40         node->right = insert_node(node->right, data); // 오른쪽에 저장한다
41
42     return node;
43 }
44
```

```

45 // 노드의 총 개수를 파악하는 함수
46 int get_node_count(TreeNode *node)
47 {
48     int count = 0;
49
50     if (node != NULL)
51     {
52         count = 1 + get_node_count(node->left) + get_node_count(node->right); // 노드 1 + 재귀 함수를 통해 좌 우의 개수를 파악해 count에 저장한다
53     }
54     return count;
55 }
56
57 // 완전 이진트리 판단 함수
58 int perfect(TreeNode *root, int in, int cnt)
59 {
60
61     if (root == NULL) // 만약 현재 위치에 아무것도 없을 경우는
62         return 1; // 참
63
64     if (in >= cnt) // 만약 현재 위치의 인덱스가 노드의 크기보다 크거나 같으면
65         return 0; // 거짓
66
67     // 재귀함수를 진행하면서 둘 중 하나라도 거짓이 나올 경우는 완전 이진트리에서 어긋나기 때문에 참 거짓을 판별하는 조건문
68     if (perfect(root->left, in + 2 + 1, cnt) && perfect(root->right, in + 2 + 2, cnt))
69         return 1;
70     else
71         return 0;
72
73 }
74
75
76
77 int main(void)
78 {
79     TreeNode *root = NULL; // 루트 노드를 생성한다
80     FILE *fp; // 파일 포인터 fp를 선언한다
81
82     int num; // 데이터 임시 저장소
83     int cnt; // 총 노드 개수 저장
84
85     fp = fopen("data1.txt", "r"); // 파일을 오픈한다
86
87     if (fp == NULL) // 만약 오픈 실패시 종료
88     {
89         printf("파일 오픈 실패");
90         return 0;
91     }
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112

```

```

77 int main(void)
78 {
79     TreeNode *root = NULL; // 루트 노드를 생성한다
80     FILE *fp; // 파일 포인터 fp를 선언한다
81
82     int num; // 데이터 임시 저장소
83     int cnt; // 총 노드 개수 저장
84
85     fp = fopen("data1.txt", "r"); // 파일을 오픈한다
86
87     if (fp == NULL) // 만약 오픈 실패시 종료
88     {
89         printf("파일 오픈 실패");
90         return 0;
91     }
92
93     // 파일이 끝날때 까지 반복한다
94     while (!feof(fp))
95     {
96         fscanf(fp, "%d ", &num); // 파일에서 읽어 와서 저장을 한다
97         printf("Inserted %d\n", num);
98         root = insert_node(root, num); // root 노드에 값을 넣는다
99     }
100
101     cnt = get_node_count(root); // 총 노드 개수 함수를 호출한다
102
103     if (perfect(root, 0, cnt)) // 판단 함수에 트리, 노드의 총 개수, 인덱스 처음의 위치를 넣어준다. 루트 노드의 인덱스는 0으로 시작한다.
104         printf("완전 이진트리입니다\n");
105     else
106         printf("완전 이진트리가 아닙니다\n");
107
108     fclose(fp);
109     free(root);
110     return 0;
111 }
112

```

2.3 소스 코드 분석

```
// 필요한 헤더파일을 선언한다
#include <stdio.h>
#include <stdlib.h>
```

1. 필요한 헤더 파일을 선언한다

```
// 트리 구조체를 만든다
typedef struct TreeNode
{
    int key; // 트리의 노드 값
    struct TreeNode *left, *right; // 좌 우 트리 연결
}TreeNode;
```

2. 트리 구조체를 만들어 준다.

3. 트리의 노드 값을 넣을 수 있는 int key 변수를 선언하고 좌 우 트리를 연결할 수 있는 자기 참조 링크를 만들어준다.

```
// 새로운 노드 생성 함수
TreeNode * new_node(int data)
{
    TreeNode *temp = (TreeNode *)malloc(sizeof(TreeNode)); // temp 노드에 treeNode 크기만큼 동적할당 해준다
    temp->key = data; // 받아온 값을 입력한다
    temp->left = temp->right = NULL; // 노드의 좌 우는 null로 설정해준다
    return temp;
}
```

4. 새로운 노드 생성 함수를 만든다. 이 함수는 현재 노드가 null 일 경우 동적할당을 해준다.

5. 동적 할당한 노드에 받아 온 값을 넣는다.

6. 트리 노드여서 좌 우 링크가 생겼기 때문에 null 로 설정해준다.

```

// 노드 삽입 함수
TreeNode * insert_node(TreeNode * node, int data)
{
    if (node == NULL) // 만약 노드가 없을 경우는
        return new_node(data); // 새로운 노드를 만들어 준다

    if (node->key > data) // 현재 노드의 크기가 받은 데이터의 크기보다 크다면
        node->left = insert_node(node->left, data); // 왼쪽에 저장해준다
    else if (node->key < data) // 반대일 경우는
        node->right = insert_node(node->right, data); // 오른쪽에 저장한다

    return node;
}

```

7. 노드 삽입함수를 만든다. 만약 현재 노드가 없을 경우는 새로운 노드를 만들어 준다.

8. 현재 노드의 학번이 받은 학번이 크기보다 크다면 왼쪽 아닐경우는 오른쪽에 저장한다.

```

// 노드의 총 개수를 파악하는 함수
int get_node_count(TreeNode *node)
{
    int count = 0;

    if (node != NULL)
    {
        count = 1 + get_node_count(node->left) + get_node_count(node->right); // 노드 1 + 재귀 함수를 통해 좌 우의 개수를 파악해 count에 저장한다
    }

    return count;
}

```

9. 노드의 총개수를 파악하기 위해서는 count 변수에 각각에 서브 트리에 대해 순환 호출하고 반환되는 값에 1을 더한다.

```

// 완전 이진트리 판단 함수
int perfect(TreeNode *root, int in, int cnt)
{
    if (root == NULL) // 만약 현재 위치에 아무것도 없을 경우는
        return 1; // 참
}

```

10. 완전 이진트리 판단 함수는 우선 만약 현재 노드 위치에 아무것도 없을 경우는 참을 반환해준다.

```

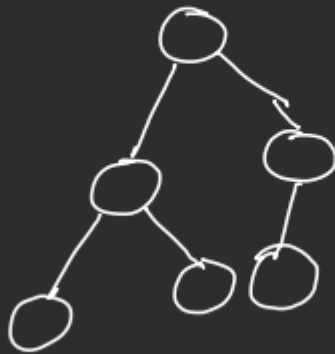
if (in >= cnt) // 만약 현재 위치의 인덱스가 노드의 크기보다 크거나 같으면
    return 0; // 거짓

// 재귀함수를 진행하면서 둘 중 하나라도 거짓이 나올 경우는 완전 이진트리에서 어긋나기 때문에 참 거짓을 판별하는 조건문
if (perfect(root->left, in + 2 + 1, cnt) && perfect(root->right, in + 2 + 2, cnt))
    return 1;
else
    return 0;

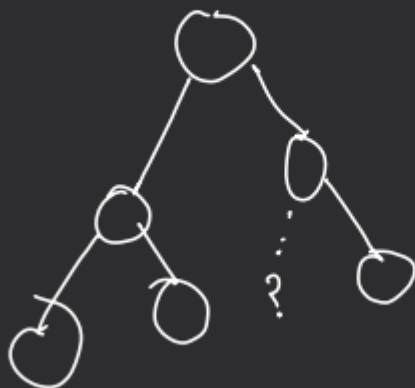
```

11. 만약 현재 위치의 인덱스가 총 노드의 크기보다 크거나 같을 경우는 거짓을 반환한다.

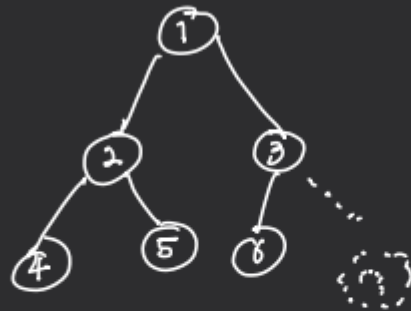
완전 이진 트리



아닌 경우



판단



총 노드의 수 = 6

1. 총 노드 수 확인

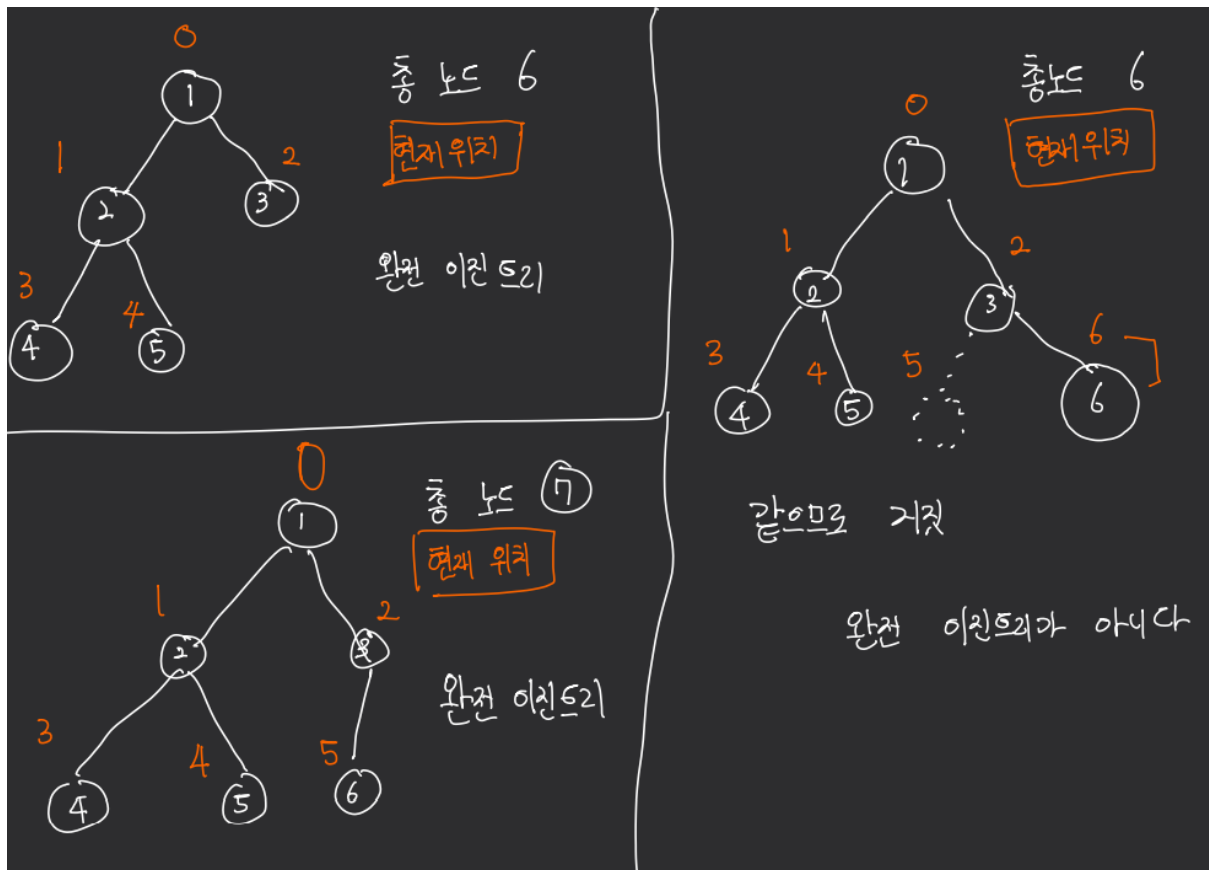
2. 처음 위치 0으로 선언,

3. 현재 노드가 NULL 이면 참 반환

(현재 노드에서 더이상 완전 이진트리가 아닐수 없음)

4. 만약 현재 위치에서 이진 트리 노드의 총 수보다
크거나 같으면 거짓 반환

5. 재귀함수로 계속 확인.



12. 두가지 모두 참일 경우는 참을 반환 하고 아닐경우는 거짓을 반환한다.

```
TreeNode *root = NULL; // 루트 노드를 생성한다
FILE *fp; // 파일 포인터 fp를 선언한다
```

```
int num; // 데이터 임시 저장소
int cnt; // 총 노드 개수 저장
```

13. 루트 노드를 생성한다

14. 학생 구조체를 생성하고 파일포인터를 선언한다.

```

fp = fopen("data1.txt", "r"); // 파일을 오픈한다

if (fp == NULL) // 만약 오픈 실패시 종료
{
    printf("파일 오픈 실패");
    return 0;
}

```

15. 파일을 오픈하고 실패할 경우 종료한다.

```

// 파일이 끝날때 까지 반복한다
while (!feof(fp))
{
    fscanf(fp, "%d ", &num); // 파일에서 읽어 와서 저장을 한다
    printf("Inserted %d\n", num);
    root = insert_node(root, num); // root 노드에 값을 넣는다
}

```

16. 파일 끝까지 반복한다

17. 파일을 읽어와서 저장 한다.

18. root 노드에 값을 넣는 insert 함수를 이용한다

```

fclose(fp);
free(root);
return 0;

```

19. 파일을 닫은 후 동적할당을 해제시켜준다


20. 종료한다.

2.4 실행 화면

 data1 - Windows 메모장


파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

8 3 10 14 2 5 9

 Microsoft Visual Studio 디버그 콘솔


```
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
완전 이진트리입니다
```

C:\Users\korca\Desktop\20204005 김필중 자료구조실습 4주차 과제\Debug\20204005 김필로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

 data1 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

8 3 10 14 2 5 9 11 16 4 6

 Microsoft Visual Studio 디버그 콘솔

```
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
Inserted 11
Inserted 16
Inserted 4
Inserted 6
완전 이진트리가 아닙니다
```

C:\Users\korca\Desktop\20204005 김필중 자료구조실습 4주차 과제\Debug로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

2.5 느낀점

이번 과제는 많이 어려운 과제였다. 노드를 넣고 하는 부분은 쉬웠으나 판단하는 부분이 많은 시도를 했으나 다 실패를 했다. 교수님께서 노드의 개수 혹은 높이를 이용하는 힌트를 주셔서 노드의 개수를 이용하기로 결정하고 많은 방법을 만들어 봤다. 노드의 개수를 가지고 위치와 비교해서 하는 방식으로 결정 한 후 완전 이진트리가 맞는지 아닌지 여러 그림을 그려보고 내가 한 방식을 대입해서 그림으로 먼저 판단을 하고 성공한 것을 확인 한 후 코드로 작성했다. 이번 과제에서 제일 중요한 것은 이진트리 판단 이었다. 이것을 해결하는 방법은 여러가지가 존재할 것이고 그 중 나는 재귀 함수를 이용해서 제작을 했다. 앞으로는 포화 상태도 있을것이고 다른 트리도 구분하는 함수를 만들어 보고 싶다.

3.1 문제 분석

이번 문제는 1번 과제에서 재귀 순환을 이용했다면 이번에는 반복을 이용할 것이다.

첫번째로는 학생 구조체를 만들어야 한다. 학생 구조체 안에는 학번을 저장하는 변수, 이름을 저장하는 변수 2개를 만들어 준다.

두번째로는 트리 구조체를 만들어 준다. 구조체에는 트리의 좌 우 링크와 학생 구조체 data를 넣어서 만들어 준다.

다음으로는 노드 삽입 함수를 만들어준다. 1번과 달리 반복을 이용할 것이기 때문에 우선 매개변수로 직접적인 노드의 주소를 받아온다. 그 후 부모노드, 현재노드, 새로운 노드 3개를 만든 후 탐색을 진행한다. 탐색에서 만약 같을 경우는 종료하고 크기를 비교해 좌 우 위치를 정한다. 동적할당을 한 후 값을 넣어주고 크기를 비교해 좌 우 부모노드와 연결해준다.

노드 순회 함수를 만들어 준다. 전위, 중위, 후위 순회 함수를 만들어서 데이터를 출력해야 한다. 그러기 위해서는 우선 재귀함수를 통해 만들어 준다. 각 순회 함수에 맞춰서 재귀 함수를 배치하는 것이 중요 포인트다. 여기서 또 추가할 사항은 마지막 데이터를 출력하는 부분일 경우 화살표를 제거해야한다. 그러기에 현재 데이터의 개수와 마지막 번째의 데이터 개수의 크기를 비교해 데이터를 출력하는 방식을 택할 것이다. 그러기에 이번에는 전역 변수를 하나 사용해야한다.

마지막으로는 노드의 개수를 세는 함수를 만들어 준다. 이 또한 재귀함수를 통해 반복을 통해 데이터의 개수를 세어 반환한다.

3. 2 소스 코드

```
1  //-----
2  // 제작 기간 : 21년 09월 22일 ~ 21년 09월 26일
3  // 제작자 : 20204005 김필중
4  // 프로그래밍 : 반복을 통한 이진트리 순회
5  //-----
6
7
8  // 필요한 헤더파일을 선언한다
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12
13 // 오류 방지 구문
14 #pragma warning (disable : 4996)
15
16 // 전역 변수 check 를 선언한다
17 int check = 0;
18
19 // student 학생 구조체를 만든다
20 typedef struct student
21 {
22     int id;
23     char name[30];
24 }Student;
25
26 // 트리 구조체를 만든다
27 typedef struct TreeNode
28 {
29     Student key; // 트리의 정보를 가지고 있는 구조체
30     struct TreeNode *left, *right; // 좌 우 트리 연결
31 }TreeNode;
32
```

```

33 // 노드 삽입 함수 반복 버전
34 void insert_node(TreeNode ** node, Student data)
35 {
36     TreeNode *p, *t; // p = 부모노드 t = 현재 위치 노드
37     TreeNode *n; // 새로운 노드
38
39     t = *node; // 노드를 받아온 후 저장
40     p = NULL; // 부모 노드는 null 값
41
42     while (t != NULL) // 만약 t가 null 이 아닐경우는
43     {
44         if (data.id == t->key.id) // 현재 데이터 구조체의 학번과 현재 위치의 노드의 학번이 같으면
45             return; // 종료
46         p = t; // 부모노드를 현재 위치로 저장
47         if (data.id < t->key.id) // 만약 데이터의 값이 t->key의 학번보다 작다면
48             t = t->left; // 왼쪽으로 이동
49         else // 아니면
50             t = t->right; // 오른쪽으로 이동
51     }
52
53     n = (TreeNode *)malloc(sizeof(TreeNode)); // 구조체의 크기만큼 동적할당
54     if (n == NULL) // 만약 n이 null 이면
55         return; // 종료
56     n->key.id = data.id; // 학번 입력
57     strcpy(n->key.name, data.name); // 문자열은 strcpy 를 통해 입력
58     n->left = n->right = NULL; // 양쪽 링크를 null로 정한다
59
60     if (p != NULL) // 만약 p가 널이 아니라면
61     {
62         if (data.id < p->key.id) // 받아온 학번과 부모의 노드 보다 작다면
63             p->left = n; // 부모 노드의 왼쪽에 노드를 저장
64         else // 아니라면
65             p->right = n; // 오른쪽에 저장
66     }
67     else *node = n; // 널이라면 node에 n을 저장
68 }
69

```



```

70 // 전위 순회 함수
71 void preorder(TreeNode *root, int cnt)
72 {
73     if (root != NULL) // 만약 노드가 널이 아닐경우는
74     {
75         if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
76         {
77             printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
78             return; // 함수 종료
79         }
80         printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
81         check++; // 전역 변수에 하나를 더해준다.
82         // 전위 순회 함수 재귀 함수를 호출한다
83         preorder(root->left, cnt);
84         preorder(root->right, cnt);
85     }
86 }
87
88 // 중위 순회 함수
89 void inorder(TreeNode *root, int cnt)
90 {
91     if (root != NULL) // 만약 노드가 널이 아닐경우는
92     {
93         // 중위 순회 함수 재귀 함수를 호출한다
94         inorder(root->left, cnt);
95         if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
96         {
97             printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
98             return; // 함수 종료
99         }
100         printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
101         check++; // 전역 변수에 하나를 더해준다.
102         // 중위 순회 함수 재귀 함수를 호출한다
103         inorder(root->right, cnt);
104     }
105 }
106
107
108
109 // 후위 순회 함수
110 void postorder(TreeNode *root, int cnt)
111 {
112     if (root != NULL) // 만약 노드가 널이 아닐경우는
113     {
114         // 후위 순회 함수 재귀 함수를 호출한다
115         postorder(root->left, cnt);
116         postorder(root->right, cnt);
117         if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
118         {
119             printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
120             return; // 함수 종료
121         }
122         printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
123         check++; // 전역 변수에 하나를 더해준다.
124     }
125 }
126
127 // 노드의 총 개수를 파악하는 함수
128 int get_node_count(TreeNode *node)
129 {
130     int count = 0;
131     if (node != NULL)
132     {
133         count = 1 + get_node_count(node->left) + get_node_count(node->right); // 노드 1 + 재귀 함수를 통해 좌 우의 개수를 파악해 count에 저장한다
134     }
135     return count; // 반환한다
136 }
137
138

```

```

140 int main(void)
141 {
142     TreeNode *root = NULL; // 루트 노드를 생성한다
143     Student s; // 구조체를 생성한다
144     FILE *fp; // 파일 포인터 fp를 선언한다
145
146     char temp[30]; // 이름 임시 저장소
147     int cnt = 0; // 총 노드 저장소
148
149     fp = fopen("data.txt", "r"); // 파일을 오픈한다
150
151     if (fp == NULL) // 오픈이 안될경우 종료한다
152     {
153         printf("파일 오픈 실패");
154         return 0;
155     }
156
157     // 파일 끝까지 반복한다
158     while (!feof(fp))
159     {
160         fscanf(fp, "%d %s\n", &s.id, temp); // 파일에서 읽어 와서 저장을 한다
161         strcpy(s.name, temp); // 문자열은 strcpy 로 저장한다
162         insert_node(&root, s); // root 노드에 값을 넣는다
163     }
164
165     cnt = get_node_count(root); // 총 노드 개수 함수를 호출한다
166
167     // 전위 함수
168     printf("전위 순회 : ");
169     preorder(root, cnt);
170     printf("\n");
171
172     // 중위 함수
173     check = 0; // 전역 변수 check를 초기화 한다
174     printf("중위 순회 : ");
175     inorder(root, cnt);
176     printf("\n");
177
178     // 후위 함수
179     check = 0; // 전역 변수 check를 초기화 한다
180     printf("후위 순회 : ");
181     postorder(root, cnt);
182     printf("\n");
183
184
185     fclose(fp); // 파일을 닫는다
186     free(root); // 동적할당 해제한다
187     return 0; // 종료한다.
188 }

```

3.3 소스 코드 분석

```
// 필요한 헤더파일을 선언한다
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 오류 방지 구문
#pragma warning (disable : 4996)
```

1. 필요한 헤더 파일을 생성한다

string.h 역할은 strcpy 를 사용하기 위함이다.

2. 오류 방지 구문을 선언한다.

```
// 전역 변수 check 를 선언한다
int check = 0;
```

3. 마지막 데이터를 출력할 때 확인하기 위한 전역변수를 선언한다.

```
// student 학생 구조체를 만든다
typedef struct student
{
    int id;
    char name[30];
}Student;
```

4. 학생 구조체를 만든다.

5. 학생 학번 변수, 이름 변수를 선언한다.

```
// 트리 구조체를 만든다
typedef struct TreeNode
{
    Student key; // 트리의 정보를 가지고 있는 구조체
    struct TreeNode *left, *right; // 좌 우 트리 연결
}TreeNode;
```

6. 트리 구조체를 만든다.

7. 트리의 정보를 가지고 있는 구조체 key를 선언한다 여기에 학생 정보를 넣어 준다.

8. 자기 참조를 통해 좌 우 링크를 만들어 준다. 이걸 통해 트리 밑으로 쪽 뺏어 나갈 수 있다.

```

// 노드 삽입 함수 반복 버전
void insert_node(TreeNode ** node, Student data)
{
    TreeNode *p, *t; // p = 부모노드 t = 현재 위치 노드
    TreeNode *n; // 새로운 노드

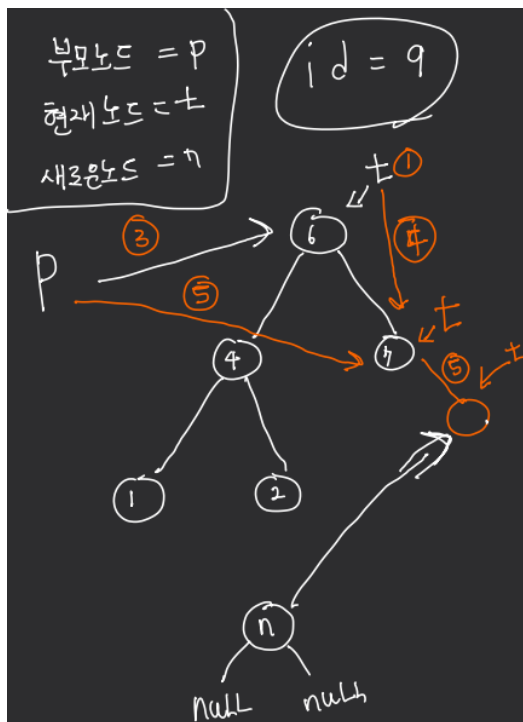
    t = *node; // 노드를 받아온 후 저장
    p = NULL; // 부모 노드는 null 값

    while (t != NULL) // 만약 t가 null 이 아닐경우는
    {
        if (data.id == t->key.id) // 현재 데이터 구조체의 학번과 현재 위치의 노드의 학번이 같으면
            return; // 종료
        p = t; // 부모노드를 현재 위치로 저장
        if (data.id < t->key.id) // 만약 데이터의 값이 t->key의 학번보다 작다면
            t = t->left; // 왼쪽으로 이동
        else // 아니면
            t = t->right; // 오른쪽으로 이동
    }

    n = (TreeNode *)malloc(sizeof(TreeNode)); // 구조체의 크기만큼 동적할당
    if (n == NULL) // 만약 n이 null 이면
        return; // 종료
    n->key.id = data.id; // 학번 입력
    strcpy(n->key.name, data.name); // 문자열은 strcpy 를 통해 입력
    n->left = n->right = NULL; // 양쪽 링크를 null로 정한다

    if (p != NULL) // 만약 p가 널이 아니라면
    {
        if (data.id < p->key.id) // 받아온 학번과 부모의 노드 보다 작다면
            p->left = n; // 부모 노드의 왼쪽에 노드를 저장
        else // 아니라면
            p->right = n; // 오른쪽에 저장
    }
    else *node = n; // 널이라면 node에 n을 저장
}

```



1. $t = *node$
2. p가 NULL이 아님
3. $p = t$
4. $data.id > t \rightarrow key.id$ 이므로
오른쪽이동 (t가)
5. 한번 더 크기 비교 해서 크므로
오른쪽 이동
6. n이 값을 삽입하고 null 값 저장
7. p가 null이 아니고 $data.id > p \rightarrow key.id$
이므로 오른쪽에 저장.

그림에 그린것처럼 삽입이 반복되면서 일어난다.

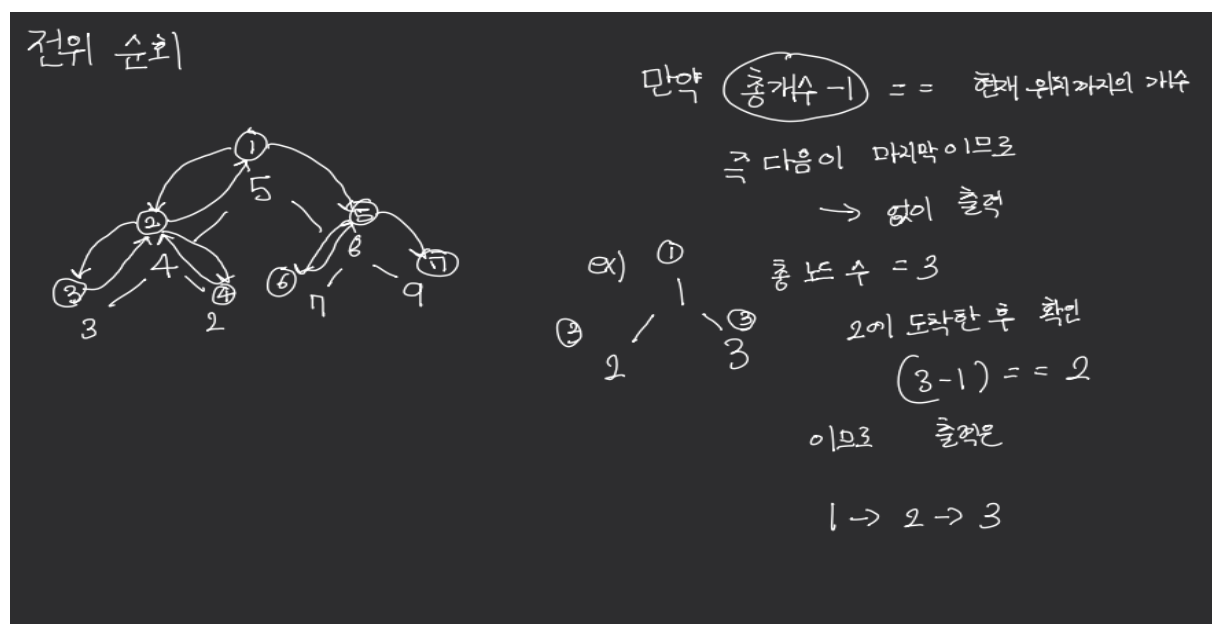
```
// 전위 순회 함수
void preorder(TreeNode *root, int cnt)
{
    if (root != NULL) // 만약 노드가 널이 아닐경우는
    {
        if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
        {
            printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
            return; // 함수 종료
        }
        printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
        check++; // 전역 변수에 하나를 더해준다.
        // 전위 순회 함수 재귀 함수를 호출한다
        preorder(root->left, cnt);
        preorder(root->right, cnt);
    }
}
```

9. 전위 순회 함수는 부모 노드부터 왼쪽 순서 다시 리턴 오른쪽으로 가는 순회이다.

10. 노드가 널이 아니고 만약 현재 노드의 개수 -1이 전체 개수랑 같을경우는 마지막 데이터 값을 화살표 없이 출력하고 종료한다.

11. 아닐 경우는 값을 출력하고 전역변수에 하나를 더해준다

12. 전위 순회 함수 재귀 함수를 루트 노드의 좌 우 값을 넣어 호출해준다.

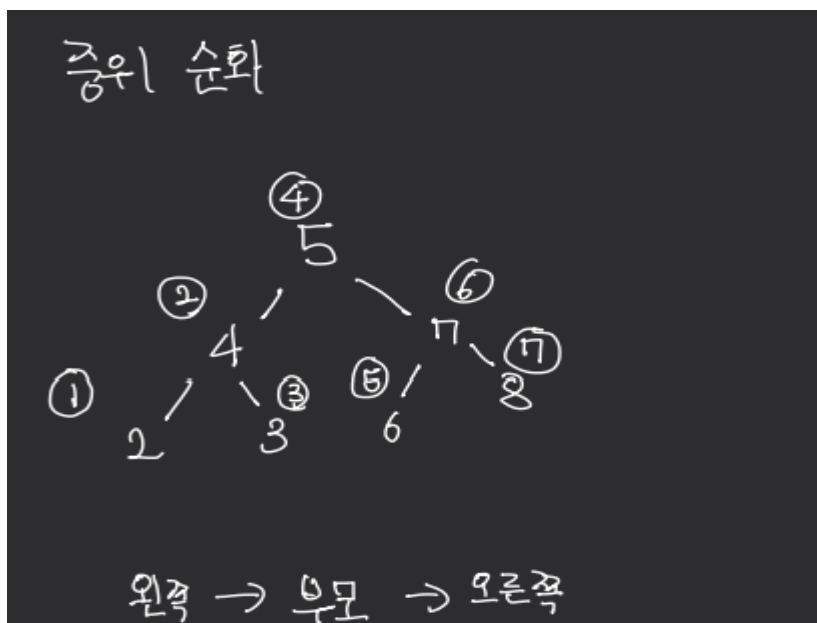


```

// 중위 순회 함수
void inorder(TreeNode *root, int cnt)
{
    if (root != NULL) // 만약 노드가 널이 아닐경우는
    {
        // 중위 순회 함수 재귀 함수를 호출한다
        inorder(root->left, cnt);
        if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
        {
            printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
            return; // 함수 종료
        }
        printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
        check++; // 전역 변수에 하나를 더해준다.
        // 중위 순회 함수 재귀 함수를 호출한다
        inorder(root->right, cnt);
    }
}

```

13. 중위 순회 함수는 왼쪽 노드부터 부모노드 오른쪽 노드로 이동하면서 프린트 한다.
14. 중위 순회 함수 재귀 함수를 루트 노드의 좌 값을 넣어 호출해준다.
15. 노드가 널이 아니고 만약 현재 노드의 개수 -1이 전체 개수랑 같을경우는 마지막 데이터 값을 화살표 없이 출력하고 종료한다.
16. 아닐 경우는 값을 출력하고 전역변수에 하나를 더해준다
17. 중위 순회 함수 재귀 함수를 루트 노드의 우 값을 넣어 호출해준다.

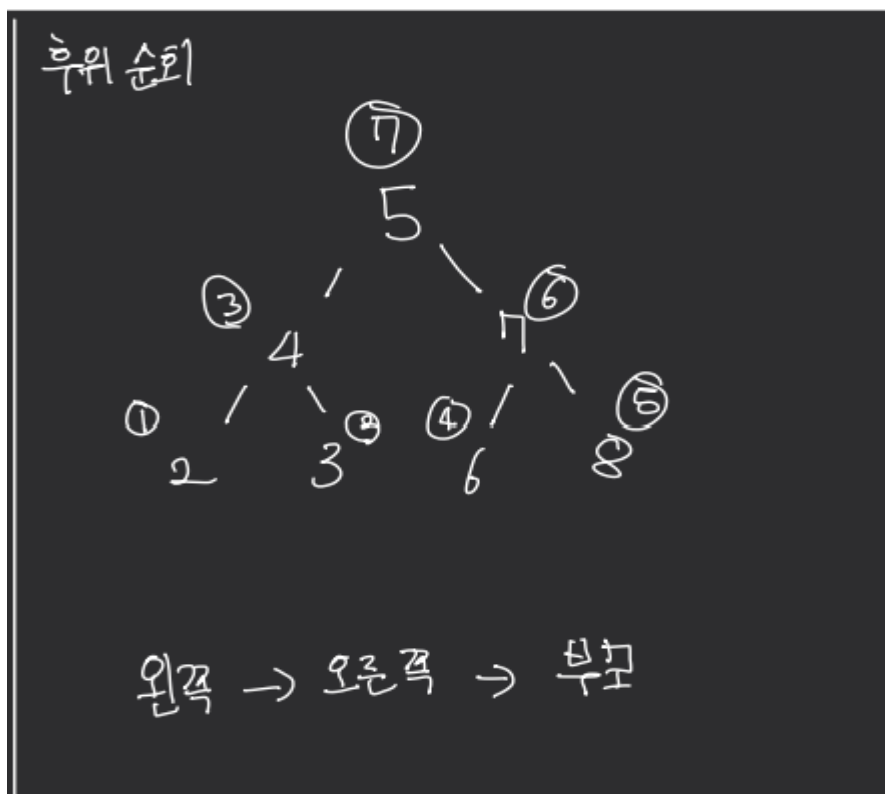


```

// 후위 순회 함수
void postorder(TreeNode *root, int cnt)
{
    if (root != NULL) // 만약 노드가 널이 아닐경우는
    {
        // 후위 순회 함수 재귀 함수를 호출한다
        postorder(root->left, cnt);
        postorder(root->right, cnt);
        if (cnt - 1 == check) // 총 노드의 개수 - 1 이 마지막 인덱스 전 위치랑 같을 경우는
        {
            printf(" %d %s", root->key.id, root->key.name); // 화살표를 제거하고 출력
            return; // 함수 종료
        }
        printf(" %d %s ->", root->key.id, root->key.name); // 값을 출력한다
        check++; // 전역 변수에 하나를 더해준다.
    }
}

```

18. 후위 순회 함수는 왼쪽노드 오른쪽 노드 부모 노드순으로 방문한다
19. 후위 순회 함수 재귀 함수를 루트 노드의 좌 우 값을 넣어 호출해준다.
20. 아닐 경우는 값을 출력하고 전역변수에 하나를 더해준다
21. 노드가 널이 아니고 만약 현재 노드의 개수 -1이 전체 개수랑 같을경우는 마지막 데이터 값을 화살표 없이 출력하고 종료한다.



```

// 노드의 총 개수를 파악하는 함수
int get_node_count(TreeNode *node)
{
    int count = 0;
    if (node != NULL)
    {
        count = 1 + get_node_count(node->left) + get_node_count(node->right); // 노드 1 + 재귀 함수를 통해 좌 우의 개수를 파악해 count에 저장한다
    }
    return count; // 반환한다
}

```

22. 노드의 총개수를 파악하기 위해서는 count 변수에 각각에 서브 트리에 대해 순환 호출하고 반환되는 값에 1을 더한다.

```

TreeNode *root = NULL; // 루트 노드를 생성한다
Student s; // 구조체를 생성한다
FILE *fp; // 파일 포인터 fp를 선언한다

char temp[30]; // 이름 임시 저장소
int cnt = 0; // 총 노드 저장소

```

23. 루트 노드를 생성한다

24. 학생 구조체를 생성하고 파일포인터를 선언한다.

```

fp = fopen("data.txt", "r"); // 파일을 오픈한다

if (fp == NULL) // 오픈이 안될경우 종료한다
{
    printf("파일 오픈 실패");
    return 0;
}

```

25. 파일을 오픈하고 실패할 경우 종료한다.


```

// 파일 끝까지 반복한다
while (!feof(fp))
{
    fscanf(fp, "%d %s\n", &s.id, temp); // 파일에서 읽어 와서 저장할 한다
    strcpy(s.name, temp); // 문자열은 strcpy 로 저장한다
    insert_node(&root, s); // root 노드에 값을 넣는다
}

cnt = get_node_count(root); // 총 노드 개수 함수를 호출한다

```

26. 파일 끝까지 반복한다

27. 파일을 읽어와서 저장할 한 후 문자열은 strcpy를 통해 값을 넣어준다

28. root 노드에 값을 넣는 insert 함수를 이용한다

29. 총 노드 개수 함수를 통해 cnt값을 저장한다

```

fclose(fp); // 파일을 닫는다
free(root); // 동적할당 해제한다
return 0; // 종료한다.

```

30. 파일을 닫고 동적할당을 해제한다

31. 종료한다,

3.4 실행 화면



The image shows two windows. The top window is a Notepad file named 'data - Windows 메모장'. It contains a list of names: 이성진, 이대호, 김철수, 루카스, 펠라인, 로토스, 토레스, and 게레로, each preceded by a number (50, 46, 44, 27, 32, 34, 71, 36 respectively). The bottom window is the Microsoft Visual Studio debugger console. It shows three lines of execution flow: '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', and '후위 순회 : 32 펠라인 -> 27 루카스 -> 44 김철수 -> 46 이대호 -> 71 토레스 -> 86 게레로 -> 84 로토스 -> 50 이성진'. Below this, it shows the file path 'D:\Users\korca\Desktop\20204005 김필중 자료구조실습 4주차 과제\Debug\20204005 김필중 자료구조실습 4주차 과제.exe(15912 프로세스)' and a message '이 창을 닫으려면 아무 키나 누르세요.'

3.5 느낀점

이번에는 1번 과제와 다르게 반복문을 사용했다. 반복의 장점은 역시 속도가 재귀함수보다 훨씬 빠르다. 속도 측정을 하지 않았지만 교수님의 예시에서 속도 차이가 난다는 것을 알게 되었다. 추후 코드를 수정에 시간을 측정해야겠다 생각을 했다. 마찬가지로 순회 함수도 반복문으로 제작을 진행하려 했으나 스택을 이용해서 진행한다는 사실을 알게 되었다. 앞으로 트리를 제작할 때 순회보다는 속도가 빠르지만 코드의 난이도가 있지만 장점이 더 좋은 반복을 위주로 제작을 해야겠다 생각을 했다.