

## 자료구조2 실습 레포트



과목명		자료구조2 실습
담당교수		홍 민 교수님
학과		컴퓨터소프트웨어공학과
학년		2학년
학번		20204059
이름		이예빈

# 목 차

---

## 1. 파일 입출력, 동적 할당과 단순 연결 리스트를 이용한 학생 정보 삽입, 삭제, 출력 프로그램

### 1.1 문제 분석

### 1.2 소스 코드

### 1.3 소스 코드 분석

### 1.4 실행창

### 1.5 느낀점

## 2. 배열로 구현하는 완전 이진 트리

### 2.1 문제 분석

### 2.2 소스 코드

### 2.3 소스 코드 분석

### 2.4 실행창

### 2.5 느낀점

## 3. 느낀점

## 1. 파일 입출력, 동적 할당과 단순 연결리스트를 이용한 학생 정보 삽입, 삭제, 출력 프로그램

### 1.1 문제 분석

#### 연결 리스트

- 파일 student.txt에 학생의 학번, 이름, 나이, 키 정보들이 저장되어 있다. 링크드 리스트에 이 값들을 저장하고 학생들의 정보를 삽입, 삭제, 출력하는 프로그램을 작성하라.
  - 파일 student.txt에 다음과 같은 형식으로 정보가 저장되어 있다
  - d 20160010 로 삭제
  - i 20120121 홍길동 95 176.2로 입력
  - p 는 모든 리스트의 데이터 출력

이 문제는 파일에 저장되어 있는 문자 정보, 그리고 학생의 정보에 따라 연결 리스트 자료에 삽입, 탐색하여 삭제, 또는 전체 데이터 출력을 하는 학생 정보 관리 프로그램을 작성하는 문제이다. 문제를 해결하기 위해 살펴보아야 할 것들은 다음과 같다.

#### 1. 파일 입출력

- 우선 파일로부터 학생의 정보를 읽기에 앞서 각 줄의 맨 처음에 등장하는 문자를 읽어 어떠한 동작을 실행할지 결정해야 한다. 문자가 i인 경우는 순서대로 학번과 이름, 나이, 키를 형식에 맞게 읽어 연결 리스트에 삽입한다. d인 경우, 학번을 입력 받고 리스트에서 해당 학번의 학생을 탐색한다. 탐색에 성공한 경우엔 해당 학생의 정보를 삭제해야 한다. p를 입력 받은 경우, 리스트의 모든 데이터를 출력해야 한다.

#### 2. 자료: 구조체, 연결 리스트

- 학생의 정보를 담을 구조체, 그리고 리스트의 노드를 구현할 구조체가 필요하다. 학생 구조체는 학번, 이름, 나이, 키를 멤버로 갖는다. 특히 이번 과제에서는 이름 멤버 변수를 문자형 포인터로 선언하여, 파일로부터 입력 받는 이름의 길이에 따라 문자열의 메모리를 동적 할당 받도록 한다. 노드 구조체에는 학생 구조체의 변수와 다음 노드를 가리킬 자체 참조 구조체 포인터를 멤버가 포함된다.

#### 3. 연결 리스트의 삽입, 검색과 삭제, 출력

- 파일로부터 읽은 문자에 따라 학생 데이터를 연결 리스트에 삽입하고, 학번을 검색하여

해당 학생의 데이터를 삭제하고, 전체 리스트의 데이터를 출력해야 한다. 연결 리스트의 삽입은 새로운 노드에 대한 동적 할당, 그리고 문자열(이름 필드)에 대한 동적 할당이 이루어진 후, 리스트의 처음에 삽입을 하게 된다. 학생 데이터 삭제의 경우, 삭제하고자 하는 학생의 학번을 입력 받고, 리스트에서 해당 학번을 가진 학생 노드를 리스트로부터의 연결 그리고 메모리를 해제한다. 리스트의 전체 데이터 출력을 위해서는, 리스트의 첫 노드부터 순회하며 각 학생의 데이터를 형식에 맞게 출력하도록 한다.

## 1.2 소스 코드

```

1  /*
2   * 작성자: .이예빈 (20204059)
3   * 작성일: .2021.09.15
4   * 프로그램: .파일에 저장되어 있는 문자와 학생 데이터를 바탕으로,
5   *           → → 연결리스트에 저장하고, 리스트의 데이터를 삭제하고, 전체 목록을 출력하는 프로그램
6   */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 // 학생 구조체
13 typedef struct Student {
14     → // 학번, 이름, 점수, 키
15     int id;
16     char *name;
17     int total;
18     double height;
19 } Student;
20
21 // 리스트의 노드 구조체
22 typedef struct ListNode {
23     → Student data;
24     → struct ListNode *link;
25 } ListNode;
26
27 ListNode *insert_first(ListNode *head, ListNode *new_node);
28 ListNode *search_delete(ListNode *head, int key);
29 void print_list(ListNode *head);
30 void clear(ListNode *head);
31 void error(char *message);
32
33 int main() {
34     → FILE *fp;
35     → ListNode *head = NULL, *new_node = NULL; // 리스트 head 포인터, 새 노드
36     → Student tmp; // 임시 학생 구조체
37     → int id; // 삭제하고자 하는 학생의 학번
38     → char name[20]; // 임시 문자열
39     → char menu; // i: insert, d: 삭제, p: 리스트 출력
40
41     → fp = fopen("data.txt", "rt");
42     → if (fp == NULL) error("파일 열기 실패");
43
44     while (!feof(fp)) {
45         → fscanf(fp, "%c", &menu); // 동작할 메뉴 값 읽기
46
47         if (menu == 'i') { // 학생 데이터 삽입
48             → fscanf(fp, "%d%s%d%lf", &tmp.id, &tmp.name, &tmp.total, &tmp.height);
49
50             → ListNode *new_node = (ListNode *) malloc(sizeof(ListNode)); // 새로운 노드 동적 할당
51             → if (new_node == NULL) error("새로운 노드 동적 할당 실패");
52             else {
53                 → new_node->data = tmp; // 데이터 필드에 tmp 구조체 대입
54                 → new_node->data.name = (char *) malloc(sizeof(name)); // name 데이터 필드에 메모리 동적 할당
55                 → strcpy(new_node->data.name, name); // name 문자열을 name 필드에 복사
56
57                 → head = insert_first(head, new_node); // 생성된 새로운 노드를 리스트의 처음에 삽입
58             }
59         }
60         else if (menu == 'd') { // 학번 검색하여 해당 학생 정보 삭제
61             → fscanf(fp, "%d", &id); // 탐색할 id 값
62             → head = search_delete(head, id);

```

```

63     }
64     else if (menu == 'p') → // 리스트 전체 출력
65     {
66         print_list(head);
67     }
68     clear(head); → // 메모리의 전체 노드 해제
69     fclose(fp); → // 파일 닫기
70 }
71
72 void error(char* message) {
73     fprintf(stderr, "%s\n", message);
74     exit(1);
75 }
76
77 // 새로운 노드를 리스트의 처음에 삽입
78 ListNode* insert_first(ListNode* head, ListNode* new_node) {
79     new_node->link = head; → // 새로운 노드의 링크 필드에 헤드 포인터값 대입
80     head = new_node; → // 다시 head에는 새로운 노드의 주소 대입
81     printf(">> %s(%d) 의 학생의 모든 정보가 삽입되었습니다.\n", new_node->data.name, new_node->data.id);
82     return head;
83 }
84
85 void print_list(ListNode* head) {
86     ListNode* p = head;
87     int cnt = 1; → // 목록의 인덱스는 1부터 시작
88
89     printf("\n===== 학생 리스트 출력 =====\n");
90     while (p != NULL) {
91         printf("%2d.%10d.%20s.%3d.%5.11f\n", cnt++, p->data.id, p->data.name, p->data.total, p->data.height);
92         p = p->link;
93     }
94     printf("=====\n\n");
95 }
96
97 // 학번 탐색하여 삭제
98 ListNode* search_delete(ListNode* head, int key) {
99     ListNode* p = head, *prev = head;
100
101     while (p != NULL) {
102         // 전체 노드 순회하며, 노드의 id 데이터 필드가 key 값과 동일한 경우를 탐색
103         if (p->data.id == key) {
104             if (p == head) head = p->link; → 첫 노드인 경우에는 head에는 두번째 노드의 주소 삽입
105             else prev->link = p->link; → 다음 노드를 이전 노드와 연결
106             printf(">> %s(%d) 의 학생의 모든 정보가 삭제되었습니다.\n", p->data.name, p->data.id);
107             free(p);
108             return head; → 노드 삭제 후 헤드 포인터 반환
109         }
110         prev = p; → 이전 노드를 현재 노드로 변경
111         p = p->link; → 다음 노드로 이동
112     }
113     // 리스트 전체 순회 후 탐색 실패하는 경우
114     printf("학번 %d 탐색 실패\n", key);
115     return head;
116 }
117
118 void clear(ListNode* head) {
119     ListNode* p = head, *next;
120     while (p != NULL) {
121         next = p->link;
122         free(p);
123         p = next;
124     }

```

### 1.3 소스 코드 분석

1. 작성자, 작성일, 프로그램명을 주석에 포함하여 작성한 뒤, 필요한 헤더 파일들을 추가한다. 표준 입출력과 동적 할당에 필요한 표준 라이브러리, 문자열 처리를 위한 string 헤더 파일들을 추가한다.

```

/*
→ 작성자: .이예빈 (20204059)
→ 작성일: .2021.09.15
→ 프로그램명: .파일에 .저장되어 .있는 .문자와 .학생 .데이터를 .바탕으로,
→ → 연결리스트에 .저장하고, .리스트의 .데이터를 .삭제하고, .전체 .목록을 .출력하는 .프로그램
*/

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

```

2. 학번(정수형), 이름(문자 포인터형), 점수(정수형), 키(실수형)를 멤버로 하는 학생 구조체 Student, 그리고 학생 구조체 변수와 다음 노드를 가리키는 자체 참조 구조체 포인터 변수를 멤버로 하는 리스트의 노드 구조체 ListNode 를 선언한다.

```

// .학생 .구조체
typedef struct Student {
→ // .학번, .이름, .점수, .키
→ int id;
→ char *name;
→ int total;
→ double height;
} Student;

// .리스트의 .노드 .구조체
typedef struct ListNode {
→ Student data;
→ struct ListNode *link;
} ListNode;

```

3. 프로그램에서 쓰이는 함수들의 원형을 선언한다. 오류가 발생한 경우 오류 메시지를 출력하고 프로그램 전체를 종료하는 error, 리스트의 처음에 새로운 노드를 삽입하는 insert\_first, 학생의 학번을 리스트에서 검색하여 해당 학생의 정보를 삭제하는 search\_delete, 리스트의 전체 데이터를 출력하는 print\_list, 리스트의 전체 노드의 메모리를 해제하는 clear 함수들이다.

```

void error(char *message);
ListNode *insert_first(ListNode *head, ListNode *new_node);
ListNode *search_delete(ListNode *head, int key);
void print_list(ListNode *head);
void clear(ListNode *head);

```

main 함수 시작 전에, 각 함수들을 살펴보자.

4. 연결 리스트의 head 포인터와 새로운 데이터를 담은 노드를 매개 변수로 전달받아 리스트의 처음에 새로운 노드를 삽입하는 insert\_first 함수이다. 새로운 노드의 링크 필드에는 head 포인터 값을 대입하고, 그리고 다시 head 에는 이 새로운 노드의 주소를 대입한다. 따라서 이전의 head 가 가리키던 노드가 현재 새로운 노드의 다음 노드가 되고, 새로운 노드가 head 의 자리에 오게 되는 것이다. 이후 출력문이 실행되고, 변경된 head 포인터가 반환된다.

```
//새로운.노드를.리스트의.처음에.삽입
ListNode* insert_first(ListNode* head, ListNode* new_node) {
    → new_node->link = head; → //새로운.노드의.링크.필드에.헤드.포인터값.대입
    → head = new_node; → //다시.head에는.새로운.노드의.주소.대입
    → printf(">> %s(%d)의.학생의.모든.정보가.삽입되었습니다.\n", new_node->data.name, new_node->data.id);
    → return head;
}
```

5. 전체 리스트의 데이터를 출력하는 print\_list 함수이다. 포인터 p가 head를 가리키고, 리스트의 첫 노드부터 순회하면서 각 노드의 데이터를 출력하게 된다. cnt 정수 값은 1로 초기화되어 있고, 각 노드를 순회할 때마다 그 값이 1씩 증가한다. 문제의 실행창 예시와 같이 형식에 맞게 출력하기 위해서 공백과 소수점 아래의 숫자 길이를 고려하여 형식 지정자를 설정하였다.

```
void print_list(ListNode* head) {
    → ListNode* p = head;
    → int cnt = 1; → //목록의.인덱스는.1부터.시작

    → printf("\n=====학생.리스트.출력===== \n");
    → while (p != NULL) {
        → printf("%2d.%10d.%20s.%3d.%5.1lf\n", cnt++, p->data.id, p->data.name, p->data.total, p->data.height);
        → p = p->link;
    }
    → printf("===== \n\n");
}
```

6. 리스트의 head 포인터, 그리고 삭제하고자 하는 학생의 학번인 key를 매개 변수로 전달받아 리스트에서 해당 학번을 가진 학생을 탐색한 후, 탐색에 성공하면 해당 데이터를 삭제하는 search\_delete 함수이다.

```
//학번.탐색하여.삭제
ListNode* search_delete(ListNode* head, int key) {
    → ListNode* p = head, *prev = head;

    → while (p != NULL) {
        → //전체.노드.순회하며, .노드의.id.데이터필드가.key값과.동일한.경우를.탐색
        → if (p->data.id == key) {
            → if (p == head) → head = p->link; → //첫.노드인.경우에는.head에는.두번째.노드의.주소.삽입
            → else prev->link = p->link; → //다음.노드를.이전.노드와.연결
        }
    }
}
```

7. 우선, ListNode형 포인터 p와 prev는 head의 주소 값을 갖도록 초기화 되어있다. 포인터 p는 리스트의 전체 노드를 순회하는 포인터이고, prev는 포인터 p가 가리키는 노드의 바로 이전 노드



를 가리키며, p가 가리키는 노드가 key의 학번과 동일하여 삭제되는 경우, 끊어지는 prev의 링크 필드를 p가 가리키는 노드의 다음 노드와 연결하기 위해 필요한 포인터이다.

8. while문 내에서는 p가 리스트의 전체 노드를 차례대로 순회하며 data 구조체의 id 필드 값이 key와 동일한지 검사를 한다. 각 노드를 순회하며, prev 포인터 또한 p를 따라가게 된다.

삭제하고자 하는 학생 데이터를 발견하여 탐색에 성공한 경우, 우선적으로 링크를 변경하는 절차를 거친다. 만약 첫번째 노드의 데이터가 삭제하고자 하는 데이터라면, head는 두번째 노드(p->link)의 주소 값을 가지게 된다. 첫번째 노드가 삭제할 대상이 아닌 경우라면, 이전 노드의 주소의 링크 필드에는 p가 가리키는 노드의 링크 필드 값을 대입하여, 삭제할 노드의 이전 노드와 다음 노드를 연결한다.

9. 링크 필드의 값을 변경했다면, 이제 탐색에 성공한 노드의 메모리를 해제할 차례이다. 우선 탐색에 성공했으므로 우선적으로 학생의 정보가 삭제되었다는 출력문을 띄우도록 하고, free 함수를 이용하여 p가 가리키는 노드의 메모리를 해제하도록 한다. 이렇게 삭제를 마친 후에는, 헤드 포인터를 반환하여 해당 함수의 실행을 종료한다.

```
→ → → printf(">>·%s(%d)의·학생의·모든·정보가·삭제되었습니다·\n",·p->data.name,·p->data.id);
→ → → free(p);
→ → → return·head; → //·노드·삭제·후·헤드·포인터·반환
→ → }
→ → prev·:=·p; → → //·이전·노드를·현재·노드로·변경
→ → p·:=·p->link; → //·다음·노드로·이동
→ }
→ //·리스트·전체·순회·후·탐색·실패하는·경우
→ printf("학번·%d·탐색·실패\n",·key);
→ return·head;
→ }
```

10. 8번에서 설명한 바와 같이, p 포인터가 하나의 노드를 거친 이후에는 prev는 해당 p의 주소 값을 가지게 되고, p는 다음 노드를 거치기 위해 다음 링크 필드 값을 가지게 된다.

11. 만약, 리스트를 전체 순회하여 while 반복문이 종료되었는데도 함수가 종료되지 않은 경우는 탐색에 실패한 경우이다. 따라서 과제 예시 실행창에는 나타나 있지 않지만, 탐색 실패한 경우의 출력문을 따로 작성해 보았다. 출력문이 실행되고 이전과 동일한 head 포인터 값이 반환된다.

12. 마지막으로 head 포인터를 매개 변수로 받아, 전체 리스트의 노드의 메모리를 해제하는 clear 함수이다. ListNode형 포인터 p, 그리고 next가 리스트 전체 노드를 순회하며 차례대로 메모리를 해제한다. next는 다음 노드를 가리키고, p가 가리키는 노드의 메모리를 해제한다.

```

void clear(ListNode* head) {
    → ListNode* p = head, *next;
    → while (p != NULL) {
    →     next = p->link;
    →     free(p);
    →     p = next;
    → }
}

```

이제 main함수를 살펴봄으로써, 파일 입출력, 그리고 각 함수들의 호출 과정을 통해 어떻게 프로그램이 동작하는지 알아보자.

13. 파일 포인터 fp, 학생 데이터를 저장할 연결 리스트를 가리킬 ListNode형 포인터 head와 새로운 노드의 주소를 가리키는 포인터 new\_node, 파일로부터 학생 데이터를 읽어 임시로 저장할 Student형 구조체 변수 tmp, 삭제할 학생의 학번을 입력 받을 정수형 변수 id, 파일로부터 학생의 이름 데이터를 읽어 임시로 문자열을 저장해 놓을 길이가 20인 name 배열, 그리고 리스트에 노드 삽입, 리스트로부터 데이터 검색과 삭제, 그리고 출력의 동작을 결정할 문자를 읽을 menu를 선언한다.

```

int main() {
    → FILE* fp;
    → ListNode* head = NULL, *new_node = NULL; //리스트.head.포인터, 새.노드
    → Student tmp; //임시.학생.구조체
    → int id; //삭제하고자.하는.학생의.학번
    → char name[20]; //임시.문자열
    → char menu; //i:insert, d:삭제, p:리스트.출력
}

```

14. 학생 데이터들이 저장되어 있는 data.txt 파일을 읽기 모드로 연다. 파일 열기를 실패한 경우의 코드도 포함한다. 파일 읽기가 성공한 경우, 파일을 처음부터 끝까지 읽기 시작한다. 우선 각 줄에 첫번째로 저장되어 있는 문자를 menu 변수를 통해 먼저 읽고, menu의 값에 따라 동작을 결정하도록 한다.

```

    → fp = fopen("data.txt", "rt");
    → if (fp == NULL) {
    →     printf("파일 열기 실패\n");
    →     exit(1);
    → }

    → while (!feof(fp)) {
    →     fscanf(fp, "%c", &menu);
    →     //파일의 데이터.마지막에는 enter.삽입되어야 함
    → }

```

15. 우선 menu의 값이 i인 경우는 학생의 데이터를 담은 노드를 동적 할당하여 insert\_first 함수를 호출하고, 해당 노드를 리스트의 처음에 삽입하게 된다.

차례대로 학생의 학번(정수형), 이름(문자 배열), 점수(정수형), 키(실수형)을 각각 파일에서 읽는다. ListNode형 포인터 new\_node에 ListNode의 크기만큼 메모리를 동적으로 할당 받고, 실패한 경우에는 문자열을 error 함수의 인수로 전달하여 프로그램을 종료하도록 한다. 성공한 경우, 우선 new\_node의 데이터 필드에 tmp 구조체를 대입한다. 이후 데이터 필드의 name 포인터에 파일에서 읽은 name 문자 배열의 길이만큼 메모리를 할당 받고, 해당 메모리의 주소에 strcpy 함수를 통해 문자열을 복사하여 저장한다.

이로써 new\_node에 학생 구조체의 모든 데이터가 저장되고, insert\_first 함수를 호출하여 새로운 노드를 리스트의 처음에 삽입하도록 한다. insert\_first함수는 위에서 설명된 것과 같이, 노드의 링크 필드를 리스트와 연결한다.

```

→ if(menu=='i'){ → //학생.데이터.삽입
→ fscanf(fp,"%d.%s.%d.%lf",&tmp.id,&name,&tmp.total,&tmp.height);
→
→ ListNode*new_node==(ListNode*)malloc(sizeof(ListNode)); → //새로운.노드.동적.할당
→ if(new_node==NULL)error("새로운.노드.동적.할당.실패");
→ else{
→ new_node->data=tmp; → //데이터.필드에.tmp.구조체.대입
→ new_node->data.name=(char*)malloc(sizeof(name)); → //name.데이터.필드에.메모리.동적.할당
→ strcpy(new_node->data.name,name); → //name.문자열을.name.필드에.복사
→
→ head=insert_first(head,new_node); → //생성된.새로운.노드를.리스트의.처음에.삽입
→ }
→ }

```

16. menu의 값이 d인 경우에는, 삭제하고자 하는 학번을 파일로부터 읽어 해당 학생의 데이터를 삭제하도록 한다.

menu 값 다음으로 저장되어 있는 id 정수 값을 파일로부터 읽고, search\_delete 함수에 인수로 전달하여 해당 함수 내에서 id 값에 해당하는 학생의 데이터를 탐색하도록 한다.

```

→ else if(menu=='d'){ → //학번.검색하여.해당.학생.정보.삭제
→ fscanf(fp,"%d",&id); → //탐색할.id값
→ head=search_delete(head,id);

```

17. 마지막으로, 메뉴 값이 p인 경우에는 head 포인터 값을 인수로 전달하여 print\_list 함수를 호출하고 리스트의 전체 학생 데이터를 출력하도록 한다.

파일의 끝에 도달한 후 반복문이 종료된 후에는, clear 함수를 호출하여 리스트의 전체 노드의 메모리를 해제한다. 마지막으로 파일을 닫고 전체 프로그램은 실행을 종료하게 된다.

```
→ → else.if(menu=='p') → → //리스트.전체.출력
→ → → print_list(head);
→ }

→ clear(head); → //리스트의.전체.노드.메모리.해제
→ fclose(fp); → → //파일.닫기
}
```

## 1.4 실행창

### data.txt 파일

다음과 같이 각 줄에 리스트와 관련된 동작을 결정할 문자(i, p, 또는 d)와 각 동작에 따라 동일 한줄에 데이터가 각각 다르게 저장되어 있다.

```
data.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
i 20120121 홍길동 95 176.2
i 20210011 아스트라제네카 23 156.3
i 20200151 안센 27 188.3
i 20190171 화이자 34 198.2
i 20170234 이수근 42 143.9
p
d 20170234
p
i 20140177 박찬호 40 189.2
i 20100133 현주엽 43 207.3
d 20200151
i 20210154 류현진 37 187.9
p
```

### 실행 결과

```
Microsoft Visual Studio 디버그 콘솔
>> 홍길동(20120121)의 학생의 모든 정보가 삽입되었습니다.
>> 아스트라제네카(20210011)의 학생의 모든 정보가 삽입되었습니다.
>> 안센(20200151)의 학생의 모든 정보가 삽입되었습니다.
>> 화이자(20190171)의 학생의 모든 정보가 삽입되었습니다.
>> 이수근(20170234)의 학생의 모든 정보가 삽입되었습니다.

===== 학생 리스트 출력 =====
1 20170234 이수근 42 143.9
2 20190171 화이자 34 198.2
3 20200151 안센 27 188.3
4 20210011 아스트라제네카 23 156.3
5 20120121 홍길동 95 176.2
=====

>> 이수근(20170234)의 학생의 모든 정보가 삭제되었습니다.

===== 학생 리스트 출력 =====
1 20190171 화이자 34 198.2
2 20200151 안센 27 188.3
3 20210011 아스트라제네카 23 156.3
4 20120121 홍길동 95 176.2
=====

>> 박찬호(20140177)의 학생의 모든 정보가 삽입되었습니다.
>> 현주엽(20100133)의 학생의 모든 정보가 삽입되었습니다.
>> 안센(20200151)의 학생의 모든 정보가 삭제되었습니다.
>> 류현진(20210154)의 학생의 모든 정보가 삽입되었습니다.

===== 학생 리스트 출력 =====
1 20210154 류현진 37 187.9
2 20100133 현주엽 43 207.3
3 20140177 박찬호 40 189.2
4 20190171 화이자 34 198.2
5 20210011 아스트라제네카 23 156.3
6 20120121 홍길동 95 176.2
=====

C:\Users\#이예빈\Desktop\CSE 2021-2\자료구조2\code files\#20204059_실습_
료되었습니다(코드: 0개).
```

### 1.5 느낀점

그동안 2 주동안 포인터, 구조체, 파일 입출력, 연결 리스트를 복습하며 풀던 과제들과 차이점이 있다면, 파일에 리스트에 저장할 모든 데이터가 같은 형식으로 저장되어 있지 않았다는 점이다. 이번 과제에서는 학생 데이터를 리스트에 삽입할지, 리스트로부터 탐색하여 삭제할지, 혹은 출력할지에 대한 동작이 문자 형태로 파일에 저장되어 있어 각 동작에 따른 데이터도 파일에 각기 다른 형태로 저장되어 있었다. 그렇기 때문에 파일로부터 데이터를 읽을 때 우선적으로 문자를 먼저 읽어 문자가 무엇인지에 따라 각기 다른 함수를 호출하여 서로 다른 동작을 실행하도록 프로그램을 작성해야 했다.

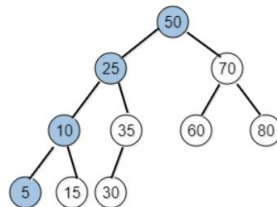
또다른 차이점은, 구조체에 문자열을 배열 형태로 멤버로 선언하지 않고, 포인터로 선언하여 프로그램 실행 도중에 문자열을 동적으로 포인터에 할당하도록 작성한 점이다. 이런 경우 이전과는 다르게, 노드를 동적으로 할당 받은 후 문자열에 대해서도 동적으로 할당 받아, 완성된 노드를 리스트에 삽입하기 위해 인수로 전달해야 했다는 점이다.

## 2. 배열로 구현한 완전 이진 트리

### 2.1 문제 분석

#### 배열로 구현하는 완전 이진트리

- 파일 data.txt에 아래와 같은 정수값들이 저장되어 있다.
- 50 25 70 10 35 60 80 5 15 30
- 배열을 이용하여 책의 p 261의 그림 7-17에 있는것과 같이 완전 이진 트리를 만든후에
- 그림에 색칠된 노드를 따라가면서 출력하는 프로그램을 작성하라.
  - 인덱스를 이용하여 현재 노드의 왼쪽 자식노드를 찾아가면서 출력하는 프로그램임

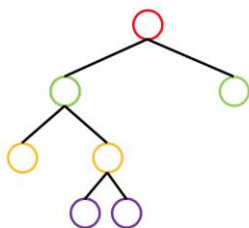


이 문제는 정수 데이터가 저장되어 있는 파일로부터 정수들을 읽어 배열로 구현된 트리 구조에 각 값들을 저장하고 루트 노드부터 시작하여 왼쪽 자식 노드를 찾아가며 출력하는 프로그램이다.

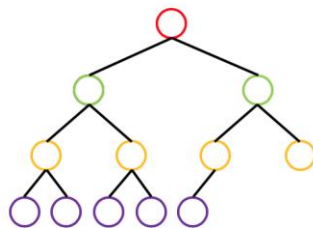
우선 이진 트리의 구조와 형태를 살펴보자.

이진 트리는 부모 노드가 최대 2 개의 자식 노드를 갖는 트리를 말한다. 이진 트리의 종류는 다음과 같은데, 이 문제에서 구현하게 되는 완전 이진 트리는, 가운데의 그림과 같이 자식 노드 중 왼쪽 자식 노드부터 먼저 채워지는 구조이며, 마지막 레벨의 자식 노드들을 제외한 모든 레벨의 노드가 모두 차 있는 형태이다.

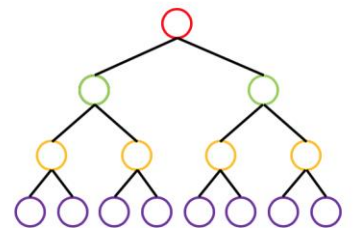
정 이진 트리 Full binary tree  
적정 이진 트리 Proper binary tree



완전 이진 트리 Complete binary tree

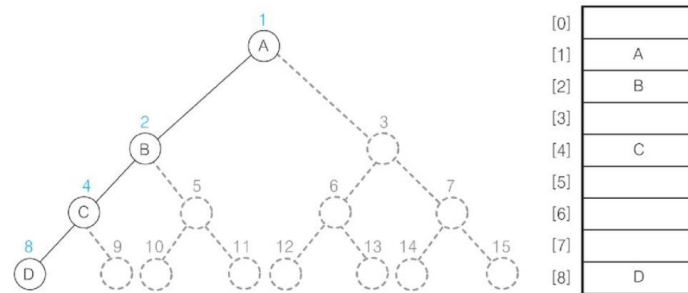


포화 이진 트리 Perfect binary tree



<https://sean-ma.tistory.com>

왼쪽 자식 노드부터 채워지며, 마지막 레벨을 제외한 다른 레벨의 모든 노드들이 가득 차 있다는 점에서 완전 이진 트리는 배열로 구현하였을 때, 다음 그림과 같이 부모 노드와 왼쪽 자식 노드, 그리고 오른쪽 자식 노드 간의 관계를 인덱스 번호로 나타내는 것이 가능하다. 루트 노드가 배열의 1 번 인덱스부터 시작하므로, 부모 노드가 인덱스  $i$  번에 저장되어 있다고 할 때, 왼쪽 자식 노드는  $2i$ , 오른쪽 자식 노드는  $2i+1$  번에 저장되어 있다고 표현할 수 있다.



따라서, 완전 이진 트리를 배열로 구현하고, 완전 이진 트리로부터 데이터를 읽을 때에는 배열 인덱스와 그 순서가 매우 중요하다. 이번 문제를 해결하기 위해서는 파일을 먼저 읽어 정수 데이터의 개수를 구하고, 그 값보다 1 더 큰 크기만큼의 정수 배열을 동적 할당하여야 한다. 두번째로 파일을 읽을 때에는, 실제 정수 데이터들을 배열에 인덱스 1 번부터 저장하게 된다. 이후, 루트 노드부터 왼쪽 자식 노드를 따라가며 데이터를 출력하기 위해 1 번 인덱스부터 2 만곱한 (2, 4, 6, 8...)번 인덱스에 저장되어 있는 데이터들을 출력하면 된다.



## 2.2 소스 코드

```

1  /*
2  →  작성자: .이예빈 (20204059)
3  →  작성일: .2021.09.15
4  →  프로그램명: .배열을 사용하여 이진 트리 구조를 완성하고,
5  →  →  →  루트 노드부터 왼쪽 자식 노드를 따라 가며 출력하는 프로그램
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 int main() {
12     FILE* fp;
13     int* treeArr = NULL;
14     int i, tmp, cnt = 1;
15
16     fp = fopen("data.txt", "rt");
17     if (fp == NULL) {
18         fprintf(stderr, "파일 읽기 실패\n");
19         exit(1);
20     }
21     // 파일로부터 읽어 전체 데이터의 개수 구하기
22     while (!feof(fp)) {
23         fscanf(fp, "%d", &tmp);
24         cnt++;
25     }
26
27     // (정수 데이터 개수 + 1) 만큼 배열 동적 할당
28     treeArr = (int*) malloc(cnt * sizeof(int));
29     rewind(fp);
30
31     // 파일 다시 읽어 트리에 저장
32     for (i = 1; i < cnt; i++)
33         fscanf(fp, "%d", &treeArr[i]);
34
35     /*
36     →  노드 i의 부모 노드 인덱스 = i / 2
37     →  노드 i의 왼쪽 자식 노드 인덱스 = 2i
38     →  노드 i의 오른쪽 자식 노드 인덱스 = 2i + 1
39     */
40
41     // 루트부터 시작하여 왼쪽 자식 노드 (2i) 를 계속 따라가며 출력하는 프로그램
42     for (i = 1; i < cnt; i *= 2)
43         printf("%d ", treeArr[i]);
44
45     free(treeArr); // 메모리 해제
46     fclose(fp); // 파일 닫기
47 }

```

## 2.3 소스 코드 분석

```

/*
  →  작성자: ·이예빈 (20204059)
  →  작성일: ·2021.09.15
  →  프로그램명: ·배열을 사용하여 이진 트리 구조를 완성하고,
  →  →  →  루트 노드부터 왼쪽 자식 노드를 따라 가며 출력하는 프로그램
*/

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

```

1. 주석에 작성자, 작성일, 프로그램명을 작성한다. 필요한 헤더 파일들인 표준 입출력, 표준 라이브러리 파일을 추가한다.

```

int main() {
  →  FILE* fp;
  →  int* treeArr = NULL;
  →  int i, tmp, cnt = 1;

  →  fp = fopen("data.txt", "rt");
  →  if (fp == NULL) {
  →  →  fprintf(stderr, "파일 읽기 실패\n");
  →  →  exit(1);
  →  }
}

```

2. main 함수의 시작 부분이다. 파일 포인터 fp, 정수 데이터들을 저장할 트리 구조의 배열을 가리키는 정수형 포인터 treeArr, 반복 제어 변수 i, 파일로부터 정수 데이터를 임시로 저장해 놓을 변수 tmp, 정수 데이터의 개수를 저장할 cnt 를 선언한다. 트리를 배열로 구현하는 경우, 첫번째 0 번 인덱스 공간은 빈 공간으로 놔두고 1 번 인덱스부터 정수를 저장하기 때문에 cnt 값은 1 로 초기화한다.

```

→  // 파일로부터 읽어 전체 데이터의 개수 구하기
while (!feof(fp)) {
  →  →  fscanf(fp, "%d", &tmp);
  →  →  cnt++;
  →  }

→  // (정수 데이터 개수 + 1) 만큼 배열 동적 할당
treeArr = (int*) malloc(cnt * sizeof(int));
rewind(fp);

```

3. 파일로부터 정수를 읽어 전체 데이터의 개수를 먼저 구하도록 한다. 읽은 정수는 tmp 에 임시로 저장하도록 하고, 반복문이 한 번 실행될 때마다 1 로 초기화되어 있던 cnt 값이 차례대로 1 씩 증가한다.

4. 파일 전체를 읽은 후에는, treeArr 포인터에 구한 cnt 의 값만큼 정수형 메모리 공간을 동적으로 할당 받는다. 다음에 다시 파일을 읽기 위해 rewind 함수를 호출하여 파일 포인터를 다시 파일의 처음으로 이동시킨다.

```
→ //파일.다시.읽어.트리에.저장
→ for(i:=1; i<cnt; i++)
→   fscanf(fp, "%d", &treeArr[i]);
```

5. 다시 파일을 읽도록 한다. 제어 변수 i가 1의 값부터 시작하여, treeArr 배열의 각 공간에 정수 데이터를 저장한다.

```
→ /*
→   → 노드.i의.부모.노드.인덱스.=i/.2
→   → 노드.i의.왼쪽.자식.노드.인덱스.=2i
→   → 노드.i의.오른쪽.자식.노드.인덱스.=2i+.1
→ */
→ //루트부터.시작하여.왼쪽.자식.노드(2i)를.계속.따라가며.출력하는.프로그램
→ for(i:=1; i<cnt; i*=2)
→   printf("%d", treeArr[i]);
```

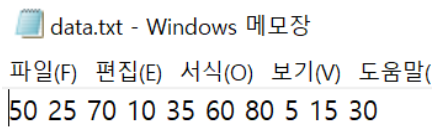
6. 트리 배열이 완성되고, 이제 루트 노드부터 시작하여 왼쪽 자식 노드를 계속 따라가며 정수 데이터들을 출력하도록 한다. 주석에 작성해 놓은 것과 같이, 왼쪽 자식 노드는 루트 노드의 인덱스의  $2i$ 에 해당하는 인덱스에 위치한다. 따라서 루트 노드가 저장되어 있는 1번 인덱스부터 시작하여, for 반복문의 증감식은 i의 값이 cnt에 도달할 때까지, 두 배로 증가하는 식이다. 각 반복이 실행될 때마다 노드의 데이터를 출력하도록 한다.

```
→ free(treeArr); → //메모리.해제
→ fclose(fp); → //파일.닫기
} □
```

7. 모든 데이터를 출력한 이후로, treeArr 포인터에 할당된 메모리를 해제하도록 하고, 파일을 닫아 전체 프로그램을 종료한다.

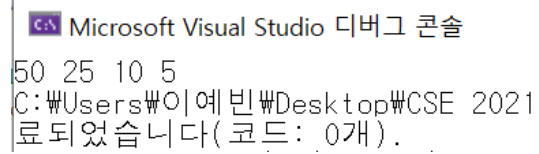
## 2.4 실행창

data.txt



data.txt - Windows 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
50 25 70 10 35 60 80 5 15 30

결과



Microsoft Visual Studio 디버그 콘솔  
50 25 10 5  
C:\Users\이예빈\Desktop\CSE 2021  
로되었습니다(코드: 0개).

## 2.5 느낀점

트리라는 자료 구조의 구조, 형태, 기본적인 용어를 비롯한 기초적인 개념들을 처음 배우고, 트리의 여러 종류 중 완전 이진 트리를 배열로 구현하는 프로그램을 작성해 보았다. 아직 트리 구조가 익숙하지는 않지만, 구조체와 리스트 이후로, 노드 간에 계층/관계를 갖는 자료 구조를 배우며 새로움을 느꼈다. 이전에는 노드와 노드가 단순히 연결되어 계층이나 노드 간의 관계를 가지지는 않았지만 트리 구조를 통해 각 노드 간의 부모/자식/형제 관계를 나타낼 수 있음을 알게 되었다. 앞으로 구조체와 같은 조금 더 복잡한 자료를 노드의 데이터 필드에 저장하게 되는 프로그램도 작성해보고 싶다.

### 3. 느낀점

---

이번 3주차 실습에서 주어진 두 문제를 풀면서 어려움은 없었다. 구조체와 연결 리스트 부분의 복습이 끝나가고, 트리 구조를 새로 배우고 있는 현 시점에서, 학생 데이터가 셀 수 없을 정도로 많아진 경우, 학번을 검색해야 하는 경우 첫번째 프로그램에서 작성한 것과 같이 첫 노드부터 순회해서 탐색하는 시간이 매우 길어질 수도 있겠다는 생각을 하게 되었다. 앞으로 트리 구조에 대해 더 배우면서 이진 트리 탐색을 비롯한 효율적인 탐색 방법을 배우고 직접 프로그램으로 작성하고 싶다.