

# 자료구조2 실습

## 6주차 과제



제출일	21.10.12	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2 실습	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

# | 목 차 |

## 1. 우선순위 큐 연습

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

## 2. 우선순위

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

## 3. 우선순위 큐를 하며 느낀 점

## 1. 우선순위 큐 연습

### 동물 히프 프로그램

---

- 배열을 이용한 히프를 사용하여 data.txt에 있는 우선 순위와 동물들의 이름을 저장하여 히프에 추가하는 프로그램을 작성하시오.

- 히프에 입력된 데이터를 히프의 루트부터 시작하여 저장되어 있는 순서대로 출력하시오

#### 1.1 문제 분석

조건 1 data.txt파일에서 데이터를 입력

조건 2 동물 앞의 번호를 이용하여 히프 정렬

조건 3 히프에 입력된 데이터를 루트부터 차례대로 출력

- 이 문제는 최대 히프를 이용한 우선순위 큐 연산으로 가장 큰 요소가 루트 노드에 저장된다. 요소는 데이터 파일에 저장된 동물이름과 함께 저장된 숫자를 이용하여 히프에 삽입한다.

히프는 완전이진트리로 만들기 때문에 배열을 이용하여 구성한다. 배열로 구성된 히프트리는 왼쪽노드로 내려가면 인덱스에 2를 곱하고 오른쪽 노드로 내려가면 2를 곱하고 1을 더하여 판단한다. 삽입을 할 때는 배열 인덱스 1번부터 차례대로 삽입하는데, 만약 중간에 더 큰 노드가 들어온다면 일단 가장 끝에 저장한 후, 부모 노드와 하나씩 비교하며 차례대로 올라간다. 그리고 만약 그 노드가 가장 크다면 루트노드가 될 것이다.

히프는 배열로 설정하는 것이 아니라 기본 Element 자료 형 포인터를 동적할당하여 선언한다. 동적할당하는 개수는 히프의 높이에 따라 달라지는데, 먼저 히프 배열을 이용하여 높이의 값을 구하고, 그 높이일 때 최대로 받을 수 있는 정도를 구하여 동적할당을 한다. n개의 높이가 주어졌을 때 최대 노드를 구하는 방법은  $2^n - 1$ 개이므로 이를 활용하여 동적할당을 한다.

히프를 출력할 때는 히프가 배열 순서대로 저장되어 있으므로 for문을 이용한 반복으로 인덱스 1부터 히프 크기까지 출력하고, 이번 문제의 경우는 삭제하는 경우가 없으므로 삽입연산까지만 구현한다. 모든 출력이 끝나면 동적할당한 문자열 포인터와 히프배열, 히프를 모두 삭제하도록 한다.

## 1.2 소스 코드

```
//
//연 : 20154018
//과 : 컴퓨터소프트웨어공학
//과 : 인문심
//과 : 학교에 입학년 대이름
//과 : 학교의 주소와 읍면리는 도로명길

//
//Define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//Define MAX_ELEMENT 200

typedef struct {
    int num; //번호
    char name; //이름 저장
}Element;
typedef struct {
    Element *head;
    int head_size; //합의 크기
}HeadType;

HeadType create(); //합을 생성하고 반환하는 함수
void init(HeadType *); //합 초기화 함수
void insert_head(HeadType *head, Element *elem); //합의 요소의 개수가 head_size인 학교에 삽입
int HeadHeight(int); //합의 높이를 반환하는 함수
void Print_head(HeadType *); //합을 출력하는 함수
void Delete_head(HeadType *); //합의 합과 요소를 제거하는 함수

int main() {
    FILE *fp; //파일 포인터
    char s[20]; //문자열을 저장하는 변수
    Element elem; //입력할 대이름을 저장 할 변수
    HeadType *head; //합
    int i; //for문을 돌 변수
    int count = 0, height; //count는 노드의 개수, height는 합과 높이

    head = create(); //합 생성
    init(head); //합 초기화
    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) {
        fscanf(fp, "%s", s, &i); //파일로부터 대이름 입력
        count++; //노드의 개수 증가
    }
    height = HeadHeight(count); //합의 높이 반환
    //합의 높이 (대이름의 개수)에 따라 노드의 개수를 증가 할 때 합과 높이를 반환 100이 넘는 노드가 있다면 새로 개수를 100으로 100이 넘어서
    head->head = (Element*)malloc(sizeof(Element)*(count + 1)); //합의 높이와 count를 반환
    head->head_size = count; //합의 높이와 count를 반환

    for(i=0; i<count; i++) {
        fscanf(fp, "%s", s, &i); //파일로부터 노드의 개수를 반환
        elem.name = (char*)malloc(sizeof(char)*(strlen(s) + 1)); //elem.name은 s를 저장
        strcpy(elem.name, s); //elem.name에 문자열 s 복사
        printf("no : %d, name : %s", i, elem.name); //입력한 대이름 출력
        insert_head(head, elem); //합에 삽입
    }
    Print_head(head); //합을 새로 출력하고 출력

    Delete_head(head); //문자열과 문자열 포인터, 합 새로, 합 요소의 제거
    fclose(fp);
    return 0;
}

HeadType create() {
    return (HeadType*)malloc(sizeof(HeadType));
}

void init(HeadType *h) {
    h->head_size = 0;
}

void insert_head(HeadType *head, Element *elem) {
    int i;
    i = h->head_size;
    //문자열의 길이를 저장하는 변수 노드의 개수
    while (i > 0) {
        if (elem->num > h->head[i-1].num) { //100이 넘는 노드는 (head)부분 노드의 개수보다 높아
            h->head[i] = h->head[i-1]; //부분 노드의 개수를 저장
            i--;
        }
        h->head[i] = *elem; //합의 크기에 head_size
    }
    int HeadHeight(int count) {
        int i, height = 0;
        for (i = 1; i <= count; i = i * 2) {
            height++;
        }
        return height;
    }

    void Print_head(HeadType *h) {
        int i;
        printf("=====문자열을 저장하고 있는 =====");
        for (i = 0; i < h->head_size; i++) {
            printf("%d : %s", i, h->head[i].name);
        }
    }

    void Delete_head(HeadType *h) {
        int i;
        for (i = 0; i < h->head_size; i++) {
            printf("no : %d", i, h->head[i].name);
            free(h->head[i].name);
        }
        free(h->head);
        free(h);
    }
}
```

## 1.3 소스 코드 분석

```
학번 : 20184612
학과 : 컴퓨터소프트웨어공학과
이름 : 김동민
파일 명: 히프에 입력된 데이터를
        히프의 루트부터 출력하는 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#define MAX_ELEMENT 200

typedef struct {
    int num;                //번호
    char *name;             //동물 이름 포인터
}Element;
typedef struct {
    Element *heap;
    int heap_size;          //힙의 크기
}HeapType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다.

2. 기본 자료형 Element를 구조체를 이용하여 선언한다. 데이터파일에는 번호와 동물 이름이 저장되어 있기 때문에 이를 입력받을 정수형 num과 문자 포인터 name을 선언한다.

3. 히프를 나타내는 HeapType을 선언한다. Element자료형으로 히프 포인터를 선언하고 높이에 맞게 동적할당하여 사용한다.

히프의 크기를 나타내는 heap\_size를 선언한다.

```
HeapType* create();           //힙을 생성하고 반환하는 함수
void init(HeapType*, int);    //힙 초기화 함수
void insert_max_heap(HeapType*, Element); //힙의 요소의 개수가 heap_size인 히프에 삽입
int Heap_Height(int);         //힙의 높이를 반환하는 함수
void Print_Heap(HeapType*);   //힙을 출력하는 함수
void Delete_Heap(HeapType*);  //힙과 힙의 포인터를 해제하는 함수
```

4. 필요한 함수를 선언한다

- create함수는 반환타입이 HeapType\*으로 히프를 생성하고 반환하는 함수이다
- init함수는 heap\_size와 heap포인터를 초기화하는 함수이다.
- insert\_max\_heap는 최대 히프의 알고리즘으로 삽입을 하는 함수이다.
- Heap\_Height함수는 완전 이진트리의 높이를 구하여 반환하는 함수이다.
- Print\_Heap함수는 반복을 이용하여 형식에 맞게 히프를 출력하는 함수이다.
- Delete\_Heap함수는 동적할당한 포인터를 해제하는 함수이다.

```

int main() {
    FILE *fp;           //파일 포인터
    char s[20];          //문자열을 입력받을 임시변수
    Element temp;        //삽입할 때 데이터를 넘길 임시변수
    HeapType * heap;     //힙
    int i;               //for문을 돌 변수
    int count = 0;       //count는 노드의 개수

    heap = create();     //힙 생성

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }

    while (!feof(fp)) { //파일 끝까지 데이터 입력
        fscanf(fp, "%d%s", &i, s);
        count++;        //노드의 개수 추가
    }

    init(heap, Heap_Height(count)); //힙 초기화
    rewind(fp);         //파일 포인터를 앞으로 옮김

    for(i=0; i<count; i++){
        fscanf(fp, "%d%s", &temp.num, s); //임시변수 temp의 num과 문자열 배열 s에 입력
        temp.name = (char*)malloc(sizeof(char)*(strlen(s) + 1)); //temp의 name을 s문자열 길이 +1로 동적할당
        strcpy(temp.name, s); //temp.name에 문자열 s 복사
        printf(">>(%d : %s) 입력\n", temp.num, temp.name); //입력된 데이터 출력
        insert_max_heap(heap, temp); //힙에 삽입
    }

    Print_Heap(heap); //힙을 배열 순서대로 출력

    Delete_Heap(heap); //동적할당한 문자열 포인터, 힙 배열, 힙 메모리 해제
    fclose(fp);       //파일 포인터 닫음
    return 0;
}

```

##### 5. 필요한 변수들을 선언한다.

- fp는 파일포인터이고 s문자열 배열은 동물의 이름을 입력받을 임시변수이다.
- temp는 Element형으로 삽입할 때 힙에 데이터를 넘길 임시변수이다.
- heap는 힙를 나타내는 변수이고 count는 노드의 개수를 나타내는 변수이다.

##### 6. 먼저 create함수를 이용하여 힙를 동적할당하여 생성한다.

파일을 열고 파일 끝까지 입력받으며 노드의 개수를 센다. 노드의 개수는 완전이진트리의 높이를 구할 때 사용한다.

##### 7. init함수를 이용하여 힙를 초기화한다. 매개 변수로는 힙 포인터와 힙의 높이를 반환하는 Heap\_Height를 전달한다. Heap\_Height함수는 노드의 개수를 전달하면 트리의 높이를 반환한다.

##### 8. rewind함수로 파일포인터를 처음으로 돌린 후 for문을 이용하여 노드의 개수만큼 다시 입력받는다. 특히 이름의 경우는 먼저 임시변수 s에 입력받은 후 temp의 name변수를 입력받은 이름의 길이 +1로 동적할당하고 strcpy함수로 복사하여 입력한다. 그리고 insert\_max\_heap으로 최대 힙에 삽입한다.

9. 삽입이 완료 되었다면 Print\_Heap함수로 힙을 반복하며 형식에 맞게 출력하고, Delete\_Heap함수를 통해 동적할당된 부분을 모두 해제해준다.  
마지막으로 fclose로 파일포인터를 닫고 프로그램을 종료한다.

```
HeapType* create() { //힙을 생성하고 반환하는 함수
    return (HeapType*)malloc(sizeof(HeapType));
}
void init(HeapType* h, int height) { //힙의 크기를 0으로 초기화
    h->heap_size = 0;
    //힙의 높이 값에 따라 비트 시프트 연산을 통해 힙 배열 동적할당
    h->heap = (Element*)malloc(sizeof(Element)*((1 << (height))));
}
```

10. create함수는 반환 타입이 HeapType\*으로 힙을 동적할당하여 반환한다.

11. init함수는 힙을 초기화하는 함수인데, 먼저 힙의 사이즈를 나타내는 heap\_size를 0으로 설정하고 heap포인터는 높이를 나타내는 변수인 height에 맞게 동적할당한다. 동적할당 할 때는 heap가 Element 자료형이기 때문에 Element 자료형으로 동적할당을 한다.

12. 동적할당하는 크기는 비트시프트연산을 통해 계산하는데, <<비트 시프트 연산은 <<뒤에 있는 숫자만큼 2를 곱하는 것과 같다. 따라서 만약 높이가 4이면 1에다가 2를 4번 곱하는 것과 같은 연산이므로 16개의 배열로 동적할당하게 된다. 높이가 4일 때 최대를 받을 수 있는 노드의 개수는  $2^4-1$ 로 1부터 15까지이기 때문에 배열은 16개의 배열로 선언한다.

13. 비트 시프트 연산을 사용하면 높이가 1일 때는 루트노드만 있기 때문에 2로 동적할당되고 2일 경우는 0,1,2,3의 인덱스를 가지기 때문에 4로 동적할당 될 것이다.



```

void insert_max_heap(HeapType*h, Element item) { //현재 요소의 개수가 heap_size 힙 h에 item을 삽입한다. (가장 큰 값이 루트)
    int i;
    i = ++(h->heap_size); //요소가 1개 늘었으므로 사이즈 1추가
    //트리틀 거슬러 올라가면서 부모 노드와 비교
    //i가 1이 아니고 입력받은 item이 부모 노드의 번호보다 크다면 반복
    while ((i != 1) && (item.num > h->heap[i / 2].num)) {
        h->heap[i] = h->heap[i / 2]; //부모 노드의 데이터를 자식 노드에 저장(비교하며 하나씩 올라감)
        i /= 2; //부모 노드로 거슬러 올라감
    }
    h->heap[i] = item; //현재 위치에 item삽입
}

```

14. insert\_max\_heap는 요소의 개수가 heap\_size인 힙에 최대 힙으로 삽입을 하는 함수이다.

먼저 I를 선언하고 가장 끝 노드의 다음 인덱스 번호를 저장한다.

15. 그리고 그 자리에 바로 item을 삽입하지 않고 item과 부모노드를 비교하며 하나씩 거슬러 올라가는데, I가 1이 아니고 입력받은 item이 부모 노드의 번호보다 크다면 반복을 수행한다.

만약 item 번호가 부모 노드의 번호보다 크다면 부모 노드의 데이터를 자식 노드의 자리에 삽입하고 I는 현재 인덱스에서 /2연산을 하여 부모 노드의 인덱스로 거슬러 올라간다.

16. 만약 I가 1이 되거나(루트노드) 부모의 데이터가 item데이터보다 더 크다면 반복을 종료하고 그 인덱스 자리에 item을 삽입한다.

```

int Heap_Height(int count) { //힙의 높이를 반환하는 함수
    int i, height = 0;
    for (i = 1; i <= count; i = i * 2) { //힙은 완전 이진트리이기 때문에 왼쪽 노드의 길이로 판단
        height++; //배열에 *2를하면 왼쪽 노드를 가리키므로 왼쪽으로 이동하며 count보다 커질때까지 반복
    } //만약 count가 8이면 4의 높이를 가짐
    return height;
}

void Print_Heap(HeapType*h) { //힙을 형식에 맞게 출력하는 함수
    int i;
    printf("\n=====동물 힙프 출력 =====\n");
    for (i = 1; i <= h->heap_size; i++) { //i를 루트노드인 1인덱스부터 힙의 크기인 heap_size까지 반복
        printf("%d: %s => ", h->heap[i].num, h->heap[i].name);
    }
}

```

17. Heap\_Height함수는 노드의 개수를 이용하여 높이를 반환하는 함수이다. for문을 1부터 노드의 개수 count까지 반복하는데 힙프는 완전이진트리이므로 왼쪽 노드로만 내려간다면 높이를 구할 수 있다.

그러므로 현재 노드에 \*2연산을 하여 인덱스가 노드 개수보다 커질 때까지 왼쪽 노드로 내려가며 height를 1씩 증가시키고 이를 반환한다.

18. Print\_Heap함수는 형식에 맞게 힙프를 출력하는 함수이다.

힙프는 인덱스 1번 자리부터 값이 있기 때문에 1부터 힙프의 size까지 반복하며 번호와 이름을 형식에 맞게 출력한다.



```

void Delete_Heap(HeapType*h) { //힙을 삭제하는 함수
    int i;
    for (i = 1; i <= h->heap_size; i++) { //동적할당한 문자열 포인터를 삭제하는 함수
        // printf("%s\n", h->heap[i].name);
        free(h->heap[i].name); //i를 루트노드인 1인덱스부터 heap_size까지 반복하며 이름 포인터 해제
    }
    free(h->heap); //동적할당한 힙 배열 해제
    free(h); //힙 해제
}

```

19. Delete\_Heap함수는 동적할당한 부분을 모두 메모리 해제해주는 함수이다.

가장 먼저 heap의 이름을 삭제한다. 0번 인덱스에는 아무 값이 없고 1번 인덱스부터 힙의 끝까지 동적할당한 이름이 존재하기 때문에 for문을 돌려 이름 포인터를 메모리 해제한다.

20. 이름포인터를 모두 해제했으면 Element형으로 동적할당한 힙 배열을 메모리 해제하고 마지막으로 힙타입인 힙을 메모리 해제한다.

## 1.4 실행 창

### - 문제형식과 데이터 파일에 맞게 출력

```
//노드의 개수 추가
//힙 초기화
//파일 포인터를 앞으
//임시변수 temp의 n
ar)*(strlen(s) + 1));
//temp.name에 문자열
num, temp.name); //인
//힙에 삽입

//힙을 배열 순서대로
//동적할당한 문자열
//파일 포인터 닫음

//힙을 생성
e));

//힙
//통해 힙 배열 동적할
int)*((1 << (height )
em) { //현재 요소의 개
//요소가 1개 늘었으
비교
노드의 번호보다 크다면
i / 2).num)) {
//부모 노드의 대
//부모 노드로 거
//현재
//힙의 높이를 반환하
//힙은 완전 이진트리

//자료구조구조2#실습#week6_1#Debug#week6_1.exe(12892
이 창을 닫으려면 아무 키나 누르세요.
```

data - Windows 메모장

13 펭귄  
7 사자  
2 표범  
15 기린  
5 고양이  
6 강아지  
9 물개  
11 다람쥐  
17 얼룩말  
1 버팔로  
20 호랑이

## 1.5 느낀 점

이번 문제는 최대 힙을 이용하여 우선순위 큐 프로그램을 만들어 삽입한 후, 이를 형식에 맞게 출력하는 과제였습니다. 책에 있는 소스코드에서는 크기를 define으로 정의하여 배열로 힙을 선언했지만 이번 과제의 경우 힙을 포인터로 선언하고 높이 값에 따라 동적할당하여 개수에 맞게 배열로 만들어 활용했습니다. 동적할당을 할 때는 높이 값이 주어졌을 때 최대를 받을 수 있는 경우로 비트시프트 연산을 사용하여 동적할당을 할 수 있었습니다.

처음 힙에 대해서 배울 땐 처음 배우는 내용이라 많이 어색하고 어려웠다고 생각했었는데 이번 과제를 프로그래밍하며 라인by라인으로 생각해보고 하나씩 거슬러 올라가는 알고리즘에 대해 이해를 할 수 있었습니다. 특히 배열로 구성했을 때 과연 전에 이진트리나 리스트로 구현했던 것보다 시간 복잡도를 줄일 수 있을까 생각했지만 이를 더 줄일 수 있는 것을 보고 신기함을 느꼈고, 항상 프로그래밍을 할 때에 그에 맞는 알고리즘을 미리 생각해보고 사용할 수 있어야 한다고 생각했습니다.

이번 과제를 통해 힙에 대해 많은 부분을 공부할 수 있었고, 앞으로는 힙을 이용하여 다른 프로그램도 도전해보고 싶다고 생각했습니다.

## 2. 우선순위 큐

### 손님 관리 프로그램

---

- data.txt에 손님의 입장과 퇴장이 이름과 함께 입력되어 있다. 각 입장과 퇴장을 힙에 적용하고 이를 힙에 저장된 순서대로 출력하시오.

#### 2.1 문제 분석

조건 1 data.txt파일에서 데이터를 입력

조건 2 I는 입장, o는 퇴장을 뜻 함

조건 3 우선 순위는 앞 사람이 퇴장하면 이름순(가나다순)으로 적용됨

- 이 문제는 최소 힙을 이용한 우선순위 큐로, data파일에 저장된 이름의 순서에 따라 가나다순으로 정렬되고 루트 노드에는 가장 작은 (가나다순으로 가장 빠른)이름이 저장되어 삽입될 것이다.

데이터 파일에는 I또는 o가 저장되어 있는데, I는 삽입연산으로 뒤에 이름이 저장되어 있어서 I가 입력된다면 뒤에 추가로 입력을 받는다. o는 삭제 연산으로 루트 노드에 있는 값을 삭제하는데, 삭제를 할 때마다 루트노드에 있는 값이 삭제되는데, 루트노드에는 가장 작은 이름이 있으므로 가나다순으로 삭제가 될 수 있다.

힙은 완전이진트리이기 때문에 배열을 이용하여 저장하는데, 이번 문제의 경우도 힙 배열을 포인터로 선언하여 동적할당하여 값을 입력받는다. 동적할당을 할 때는 삭제 연산이 없을 때를 생각하여 최대를 받을 수 있는 경우로 생각한다. 먼저 파일을 입력받으며 I의 개수를 세고 그것을 이용하여 완전이진트리의 높이를 구한 후 만약 n개의 높이일 때 최대 저장할 수 있는  $2^n - 1$ 의 개수를 이용하여 동적할당한다. 그리고 왼쪽 노드를 가리킬 때는 현재 인덱스에 2를 곱하고 오른쪽 노드를 가리킬 때는 인덱스에 2를 곱한 후 1을 더하여 노드를 찾는다.

힙을 출력할 때는 힙이 배열 순서대로 저장되어 있으므로 for문을 이용한 반복으로 인덱스 1부터 힙 크기까지 출력한다. 모든 출력이 끝나면 동적할당한 문자열 포인터와 힙배열, 힙을 모두 삭제하도록 한다.

## 2.2 소스 코드

```
/*
 * 학번 : 20184812
 * 학과 : 컴퓨터소프트웨어학과
 * 이름 : 김승민
 * 파일명 : 힙과 배열을 히프에 적용하고
 *         히프에 저장된 순서대로 출력하는 프로그램
 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char *name; //이름 포인터
} Element;

typedef struct {
    Element *heap; //힙 포인터
    int heap_size; //힙의 크기
} HeapType;

HeapType* create(); //힙을 생성하고 반환하는 함수
void init(HeapType*); //힙 초기화 함수
void insert_min_heap(HeapType* h, Element *ten); //힙의 루트의 개수가 heap_size인 히프에 삽입 (최소 히프)
Element* delete_min_heap(HeapType* h); //힙에서 루트 노드를 삭제하고 반환하는 함수
void Print_Heap(HeapType* h); //힙을 배열 순서대로 출력하는 함수
int Heap_Height(int); //힙의 높이를 반환하는 함수
void Delete_Heap(HeapType*); //힙과 힙의 포인터를 해제하는 함수

int main() {
    FILE* fp; //파일포인터
    Element temp; //삽입할때 값을 넣거줄 임시 변수
    char s[10], c; //s는 임시 문자열 변수, c는 i 또는 o를 입력받을 변수
    int count=0, height; //count는 i(삽입)의 개수, height는 높이
    HeapType *heap; //힙 타입

    heap = create(); //힙 생성
    init(heap); //힙 초기화

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open\n");
        return 0;
    }

    while (!feof(fp)) { //파일 끝까지 데이터 입력
        fscanf(fp, "%c", &c); //i 또는 o 입력
        if (c == 'i') { //만약 i를 입력할 수 있는 개수를 주함
            fscanf(fp, "%s", s);
            count++; //i의 개수 ++
        }
    }

    height = Heap_Height(count); //힙의 높이 반환
    //힙의 높이 값에 따라 메모리 공간을 통해 힙 배열 출력함
    //height가 4라면 1을 빼고 2<<3 연산 (2<<3 == 16)
    printf("i의 개수 : %d\n", count);
    printf("힙의 높이 : %d\n", height);
    printf("2 << (height-1) : %d\n", (2 << (height - 1)));
    heap->heap = (Element*)malloc(sizeof(Element)*((2 << (height-1))));
    rewind(fp); //파일포인터를 처음으로 옮김
    while (!feof(fp)) { //count에는 i의 개수만 저장되어 있기 때문에 feof함수로 끝까지 반복
        fscanf(fp, "%c", &c); //i(삽입) 또는 o(삭제) 입력받을
        switch (c) {
            case 'i': //만약 i를 입력받았다면 삽입 연산 수행
                fscanf(fp, "%s", s); //일시 문자열 배열 s에 입력받을
                temp.name = (char*)malloc(sizeof(char)*(strlen(s) + 1)); //s의 길이 +1로 temp의 name포인터 출력함
                strcpy(temp.name, s); //temp.name에 s문자열 복사
                insert_min_heap(heap, temp); //최소 히프에 삽입
                printf(">>삽입(%s) 결과\n", temp.name); //입력받은 이름 출력
                Print_Heap(heap); //히프를 알식에 맞게 출력
                break;
            case 'o': //만약 o를 입력받았다면 루트 노드의 삭제 연산 수행
                temp = delete_min_heap(heap); //삭제된 루트 노드 반환함
                printf(">>삭제(%s) 결과\n", temp.name); //삭제된 노드 출력
                free(temp.name); //삭제된 이름 포인터 메모리 해제
                Print_Heap(heap); //히프를 알식에 맞게 출력
                break;
        }
    }
    printf("\n");
    Delete_Heap(heap); //출력할때만 문자열 포인터, 힙 배열, 힙 메모리 해제
    fclose(fp); //파일 포인터 닫음
    return 0;
}
```

```

HeapType* create() {
    //힙을 생성하고 반환하는 함수
    return (HeapType*)malloc(sizeof(HeapType)); //힙 타임을 자르길에 맞게 출력할함
}

void init(HeapType* h) {
    //힙의 크기를 0으로 초기화
    h->heap_size = 0;
}

void insert_min_heap(HeapType* h, Element item) { //현재 루트의 개수가 heap_size
    int i;
    i = ++(h->heap_size);
    //트리를 거슬러 올라가면서 루트 노트와 비교
    while ((i != 1) && (strcmp(item.name, h->heap[i / 2].name) < 0)) { //트리의 루트에는 가나아 승으로 가장 빨리 나르는 이름이 들어가야 함.
        //strcmp로 단어 비교 (strcmp가 클수록 앞의 단어가 더 빠른 것)
        h->heap[i] = h->heap[i / 2]; //루트 노트의 데이터를 자식 노트에 저장
        i /= 2; //루트 노트로 이동
    }
    h->heap[i] = item; //현재 위치 인덱스에 item 데이터 저장
}

Element delete_min_heap(HeapType* h) {
    //힙에서 루트 노트를 삭제하고 반환하는 함수.
    int parent, child;
    Element item, temp;

    item = h->heap[1]; //item에 루트 노트 저장
    temp = h->heap[(h->heap_size)--]; //temp는 가장 마지막 노트를 저장하고 하나를 삭제하므로 heap_size개수 줄음
    parent = 1;
    child = 2;

    while (child <= h->heap_size) {
        //child가 히프 트리의 크기보다 작다면
        //child가 heap_size보다 작고 현재 자식 노트가 다른 자식 노트보다 가나아 승으로 늦게 나른다면(다른 자식 노트가 더 작다면) child++;
        if ((child < h->heap_size) && (strcmp(h->heap[child].name, h->heap[child + 1].name) > 0)) child++;

        //가장 작은 노트의 이름보다 현재 자식 노트보다 자식 노트가 더 늦게 나른다면 종료(temp이름이 더 작다면)
        if (strcmp(temp.name, h->heap[child].name) <= 0) break;

        h->heap[parent] = h->heap[child]; //자식 노트의 데이터를 루트 노트에 저장
        parent = child; //자식 인덱스를 루트 인덱스로 저장
        child = 2; //child에 2를 곱하여 왼쪽 노트로 내려감
    }
    h->heap[parent] = temp; //루트 노트에 temp저장(루트 노트에는 가장 작은 이름 저장됨)
    return item; //루트 노트 반환
}

void Print_Heap(HeapType* h) {
    //힙을 배열 순서로 출력하는 함수
    int i;
    printf("< 히프 출력 >\n");
    for (i = 1; i <= h->heap_size; i++) {
        printf("%d: %s => ", i, h->heap[i].name);
    }
    printf("\n");
}

int Heap_Height(int count) {
    //힙의 높이를 반환하는 함수 매개변수: 데이터 파일의 i 개수
    int i, height = 0;
    for (i = 1; i <= count; i = i * 2) {
        height++;
    }
    return height;
}

void Delete_Heap(HeapType* h) {
    //힙을 삭제하는 함수
    int i;
    for (i = 1; i <= h->heap_size; i++) {
        //출 루트노드인 인덱스 1부터 힙의 크기인 heap_size까지 반복
        //출력할때 힙의 이름 문자열 포인터를 해제
        printf("%s\n", h->heap[i].name);
        free(h->heap[i].name);
    }
    free(h->heap);
    free(h);
    //힙 배열 메모리 해제
    //힙 메모리 해제
}

```



## 2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 입력과 퇴장을 힙에 적용하고
            힙에 저장된 순서대로 출력하는 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char *name;           //이름 포인터
}Element;
typedef struct {
    Element *heap;        //힙 포인터
    int heap_size;        //힙의 크기
}HeapType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다.

2. 기본 자료형 Element를 구조체를 이용하여 선언한다. 데이터파일에는 이름이 저장되어 있기 때문에 이를 입력받을 문자 포인터 name을 선언한다.

3. 힙을 나타내는 HeapType을 선언한다. Element자료형으로 힙의 포인터를 선언하고 높이에 맞게 동적할당하여 사용한다.

힙의 크기를 나타내는 heap\_size를 선언한다.

```
HeapType* create();           //힙을 생성하고 반환하는 함수
void init(HeapType*, int);    //힙 초기화 함수
void insert_min_heap(HeapType*h, Element item); //힙의 요소의 개수가 heap_size인 힙에 삽입(최소 힙)
Element delete_min_heap(HeapType*h); //힙에서 루트 노드를 삭제하고 반환하는 함수
void Print_Heap(HeapType*h);  //힙을 배열 순서대로 출력하는 함수
int Heap_Height(int);         //힙의 높이를 반환하는 함수
void Delete_Heap(HeapType*);  //힙과 힙의 포인터를 해제하는 함수
```

4. 필요한 함수를 선언한다

- create함수는 반환타입이 HeapType\*으로 힙을 생성하고 반환하는 함수이다
- init함수는 heap\_size와 heap포인터를 초기화하는 함수이다.
- insert\_min\_heap는 최소 힙의 알고리즘으로 삽입을 하는 함수이다.
- delete\_min\_heap는 힙에서 가장 작은 노드를 반환하는 함수이다.
- Heap\_Height함수는 완전 이진트리의 높이를 구하여 반환하는 함수이다.
- Print\_Heap함수는 반복을 이용하여 형식에 맞게 힙을 출력하는 함수이다.
- Delete\_Heap함수는 동적할당한 포인터를 해제하는 함수이다.



```

int main() {
    FILE*fp; //파일포인터
    Element temp; //삽입할때 값을 넘겨줄 임시 변수
    char s[10], c; //s는 임시 문자열 변수, c는 i 또는 o를 입력받을 변수
    int count=0; //count는 i(삽입)의 개수
    HeapType *heap; //힙 타입

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("File not open\n");
        return 0;
    }
    while (!feof(fp)) { //파일 끝까지 데이터 입력
        fscanf(fp, "%c", &c); //i 또는 o 입력
        if (c == 'i') { //만약 i면 최대도 입력할 수 있는 개수를 구함
            fscanf(fp, "%s", s);
            count++; //i의 개수 ++
        }
    }
    heap = create(); //힙 생성
    init(heap, Heap_Height(count)); //힙 초기화

    rewind(fp); //파일포인터를 처음으로 옮김
    while (!feof(fp)) { //count에는 i의 개수만 저장되어 있기 때문에 feof함수로 끝까지 반복
        fscanf(fp, "%c", &c); //i(삽입) 또는 o(삭제) 입력받음
        switch (c) {
            case 'i': //만약 i를 입력받았다면 삽입 연산 수행
                fscanf(fp, "%s", s); //임시 문자열 배열 s에 입력받음
                temp.name = (char*)malloc(sizeof(char)*(strlen(s) + 1)); //s의 길이 +1로 temp의 name포인터 동적할당
                strcpy(temp.name, s); //temp.name에 s문자열 복사
                insert_min_heap(heap, temp); //최소 힙에 삽입
                printf(">>손님(%s) 입장\n", temp.name); //입력받은 이름 출력
                Print_Heap(heap); //힙을 형식에 맞게 출력
                break;
            case 'o': //만약 o를 입력받았다면 루트 노드의 삭제 연산 수행
                temp = delete_min_heap(heap); //삭제된 루트 노드 반환받음
                printf(">>손님(%s) 퇴장\n", temp.name); //삭제된 노드 출력
                free(temp.name); //삭제된 이름 포인터 메모리 해제
                Print_Heap(heap); //힙을 형식에 맞게 출력
                break;
        }
    }
    printf("\n");
    Delete_Heap(heap); //동적할당한 문자열 포인터, 힙 배열, 힙 메모리 해제
    fclose(fp); //파일 포인터 닫음
    return 0;
}

```

##### 5. 필요한 변수를 선언한다.

- fp는 파일포인터이고 s는 임시 문자열 변수, c는 명령을 입력받는다.
- temp는 Element형으로 삽입 시 값을 넘겨주고 삭제 시 받아오는 역할을 한다
- count는 I의 개수를 나타내고 heap는 힙을 나타낸다.

6. 파일을 열고 파일 끝까지 입력받으며 I의 개수를 센다. 만약 I가 입력되었다면 뒤에는 이름이 존재하므로 s로 입력을 받아주고 count를 1씩 증가시킨다.

여기서 I의 개수를 세는 이유는 만약 o로 삭제명령이 하나도 없다면 최대도 들어올 수 있는 노드의 수는 I의 개수와 같기 때문이다.

7. heap을 create함수를 통해 동적할당 생성하고, init함수로 heap의 데이터를 초기화한다. 매개 변수로는 힙 포인터와 힙의 높이를 반환하는 Heap\_Height를 전달한다. Heap\_Height함수는 노드의 개수를 전달하면 트리의 높이를 반환한다.

8. rewind함수로 파일포인터를 다시 앞으로 옮기고 feof를 통해 파일 끝까지 다시 입력받는다. 여기서 l개수로만 for문을 돌리지 않는 이유는 데이터 파일에 삭제 연산도 포함되어 있기 때문이다.

9. 먼저 fscanf로 c에 l또는 o를 입력받고 switch문을 통해 그에 맞는 동작을 한다. 만약 l를 입력받았다면 삽입연산으로 뒤에 이름이 추가로 저장되어있으므로 임시 문자열에 이를 입력받는다.

그리고 temp의 name을 문자열의 길이 +1로 동적할당하고 strcpy함수로 이를 복사한 후 최소 힙에 삽입을 한다.

삽입이 끝나면 Print\_Heap함수를 통해 삽입이 잘 되었는지 형식에 맞게 출력한다.

10. 만약 o를 입력받았다면 삭제연산으로 뒤에는 아무 값이 저장되어있지 않다. 따라서 루트노드에 있는 값을 반환하는 delete\_min\_heap함수를 호출하여 루트노드에 있는 값을 temp에 받아온다.

어떤 노드가 삭제되었는지 출력한 후, 동적할당한 이름 포인터를 free로 메모리 해제한다. 동작이 끝나면 Print\_Heap함수를 통해 삭제가 잘 되었는지 형식에 맞게 출력한다.

11. 파일 끝까지 모든 입력이 끝났다면 동적할당한 힙을 메모리 해제하는 Delete\_Heap함수를 호출한다. 마지막으로 fclose로 파일포인터를 닫고 프로그램을 종료한다.

```
HeapType* create() { //힙을 생성하고 반환하는 함수
    return (HeapType*)malloc(sizeof(HeapType)); //힙 타입을 자료형에 맞게 동적할당
}
void init(HeapType* h, int height) { //힙의 크기를 0으로 초기화
    h->heap_size = 0;
    //힙의 높이 값에 따라 비트 시프트 연산을 통해 힙 배열 동적할당.
    // printf("힙의 높이 : %d\n", height);
    // printf("(1 << (height)) : %d\n", (1 << (height)));
    h->heap = (Element*)malloc(sizeof(Element)*((1 << (height))));
}
```

12. create함수는 반환 타입이 HeapType\*으로 힙을 동적할당하여 반환한다.

13. init함수는 힙을 초기화하는 함수인데, 먼저 힙의 사이즈를 나타내는 heap\_size를 0으로 설정하고 heap포인터는 높이를 나타내는 변수인 height에 맞게 동적할당한다.

14. 동적할당하는 크기는 비트시프트연산을 통해 계산하는데, <<비트 시프트 연산은 <<뒤에 있는 숫자만큼 2를 곱하는 것과 같다. 따라서 만약 높이가 4이면 1에다가 2를 4번 곱하는 것과 같은 연산이므로 16개의 배열로 동적할당하게 된다. 높이가 4일 때 최대를 받을 수 있는 노드의 개수는  $2^4-1$ 로 1부터 15까지이기 때문에 배열은 16개의 배열로 선언한다.

```
힙의 높이 : 4
(1 << (height)) : 16
```

```
void insert_min_heap(HeapType*h, Element item) { //현재 요소의 개수가 heap_size h에 item을 삽입한다. (가장 작은 값이 루트)
    int i;
    i = ++(h->heap_size);
    //트리를 거슬러 올라가면서 부모 노드와 비교
    while ((i != 1) && (strcmp(item.name, h->heap[i / 2].name) < 0)) {
        //트리의 루트에는 가나다 순으로 가장 빨리 나오는 이름이 들어가야 함.
        //strcmp로 단어 비교 (만약 strcmp함수가 음수리턴이면 앞의 단어가 더 빠른 것), 가장 빨리 나오는 단어가 루트 노드가 됨.
        h->heap[i] = h->heap[i / 2]; //부모 노드의 데이터를 자식 노드에 저장(비교하며 하나씩 올라감)
        i /= 2; //부모 노드로 이동
    }
    h->heap[i] = item; //현재 위치 인덱스에 item 데이터 저장
}
```

15. insert\_min\_heap은 요소의 개수가 heap\_size인 힙에 최소 힙으로 삽입을 하는 함수이다. 가나다순으로 정렬하므로 루트노드에는 가장 빠른 단어가 있어야 하기 때문에 최소 힙으로 만든다.

먼저 I를 선언하고 가장 끝 노드의 다음 인덱스 번호를 저장한다.

16. 문자열의 비교는 strcmp(s1,s2)의 형식으로 비교하는데, s1이 더 빨리 나온다면 음수를 반환하고 s2가 더 빨리나온다면 양수를 반환한다.

따라서 item이름과 부모노드이름을 비교했을 때, item이 더 빨리 나온다면 노드를 거슬러 올라가야 하므로 음수가 리턴되었을 때 반복을 한다.

17. 만약 item이름이 부모 노드 이름보다 빨리 나온다면 부모 노드의 데이터를 자식 노드에 저장하고 I를 2로 나누어 I가 부모 노드의 인덱스를 가리킬 수 있도록 한다. 그리고 만약 I가 1이 되어 루트노드를 가리키거나 I가 부모 노드보다 더 늦게 나온다면 반복을 종료하도록 한다.

18. 반복이 끝났다면 현재 I가 가리키고 있는 인덱스에 item을 삽입한다. 먼저 item을 삽입하여 부모노드를 거슬러 올라가게 된다면 부모 노드와 자식노드를 바꾸는 연산도 구현 해줘야한다. 그렇기 때문에 인덱스 번호를 구하고, 가장 마지막에 그 자리에 삽입할 수 있도록 한다.

```

Element delete_min_heap(HeapType*h) {                                //힙에서 루트 노드를 삭제하고 반환하는 함수.
    int parent, child;
    Element item, temp;

    item = h->heap[1];                                              //item에 루트 노드 저장
    temp = h->heap[(h->heap_size)--];                                //temp는 가장 마지막 노드를 저장하고 하나를 삭제하므로 heap_size개수 줄임
    parent = 1;
    child = 2;

    while (child <= h->heap_size) {                                  //child가 힙 트리의 크기보다 작다면
        //child가 heap_size보다 작고 왼쪽 자식 노드가 오른쪽 자식 노드보다 가나다 순으로 늦게 나온다면(다음 자식 노드가 더 작다면) child++;
        if ((child < h->heap_size) && (strcmp(h->heap[child].name, h->heap[child + 1].name) > 0)) child++;

        //마지막 노드의 이름보다 현재 자식 노드보다 자식 노드가 더 늦게 나온다면 종료(temp이름이 더 작다면)
        if (strcmp(temp.name, h->heap[child].name) <= 0) break;

        h->heap[parent] = h->heap[child];                            //자식 노드의 데이터를 부모 노드에 저장
        parent = child;                                             //자식 인덱스를 부모 인덱스로 저장
        child *= 2;                                                 //child에 2를 곱하여 왼쪽 노드로 내려감

    }
    h->heap[parent] = temp;                                          //부모 노드에 temp저장(루트 노드에는 가장 작은 이름 저장됨)
    return item;                                                    //루트 노드 반환
}

```

19. delete\_min\_heap함수는 루트노드를 삭제하여 반환하는 함수이다.

가장 먼저 item변수에는 루트 노드를 저장하고 temp노드에는 힙의 가장 끝 노드를 저장한다. 그리고 삭제를 하므로 heap\_size의 크기를 1줄인다. parent변수에는 가장 처음 부모 노드의 인덱스인 1을 저장하고 child에는 그 부모의 왼쪽 노드인 2를 저장한다.

20. while문으로 child가 heap\_size보다 작거나 같다면 반복하고 size보다 더 커진다면 반복을 종료한다.

먼저 오른쪽 노드와 왼쪽 노드의 문자열을 비교한다.

child인덱스가 힙의 크기보다 작고, 왼쪽 자식노드가 오른쪽 자식노드보다 늦게 나올 때, 즉 strcmp함수가 양수를 리턴했을 때 오른쪽 노드의 인덱스를 가리켜야하므로 child를 1증가시킨다.

21. 이 알고리즘을 수행할 때 힙의 가장 끝의 값과 자식노드를 하나씩 비교한다. 가장 끝 값은 temp이고 이 temp이름과 현재 왼쪽 또는 오른쪽 자식의 이름을 비교한다. 여기서 음수를 리턴한다면 temp노드가 자식보다 더 빨리 나온다는 의미이므로 반복을 종료한다.

22. 만약 그렇지 않다면 자식노드의 데이터를 부모 노드에 저장하고(가장 빠른 문자열이 부모노드로 올라감) 자식 인덱스 값을 부모 인덱스 값에 저장한다. 그리고 자식은 \*2를 하여 현재 자식 노드 밑의 왼쪽 노드를 가리킨다. 이렇게 반복을 수행하고 층을 하나씩 내려가며 비교한다.

반복이 끝나면 현재 부모가 가리키는 인덱스 위치에 힙의 끝 값을 삽입하고 루트 노드를 가리키는 item을 반환한다.

```

void Print_Heap(HeapType*h) { //힙을 배열 순서로 출력하는 함수
    int i;
    printf("< 힙 출력 >\n");
    for (i = 1; i <= h->heap_size; i++) { //i를 루트노드인 인덱스 1부터 힙의 크기인 heap_size까지 반복
        printf("%d: %s => ", i, h->heap[i].name); //힙을 형식에 맞게 출력
    }
    printf("\n");
}

int Heap_Height(int count) { //힙의 높이를 반환하는 함수 매개변수: 데이터 파일의 i 개수
    int i, height = 0;
    for (i = 1; i <= count; i = i * 2) { //힙은 완전 이진트리이기 때문에 왼쪽 노드의 길이로 판단
        // 배열에 * 2를하면 왼쪽 노드를 가리키므로 왼쪽으로 이동하며 count보다 커질때까지 반복
        height++;
    }
    return height; //높이 반환
}

```

23. Print\_Heap는 배열에 저장된 힙을 형식에 맞게 반복을 이용하여 출력하는 함수이다.

힙은 인덱스 1번 자리부터 값이 있기 때문에 1부터 힙의 size까지 반복하며 번호와 이름을 형식에 맞게 출력한다.

24. Heap\_Height함수는 노드의 개수를 이용하여 높이를 반환하는 함수이다. for문을 1부터 i의 개수 count까지 반복하는데 힙은 완전이진트리이므로 왼쪽 노드로만 내려간다면 높이를 구할 수 있다.

그러므로 현재 노드에 \*2연산을 하며 인덱스가 i 개수보다 커질 때까지 왼쪽 노드로 내려가며 height를 1씩 증가시키고 이를 반환한다.

```

void Delete_Heap(HeapType*h) { //힙을 삭제하는 함수
    int i;
    for (i = 1; i <= h->heap_size; i++) { //i를 루트노드인 인덱스 1부터 힙의 크기인 heap_size까지 반복
        printf("%s\n", h->heap[i].name);
        free(h->heap[i].name); //동적할당된 힙의 이름 문자열 포인터를 해제
    }
    free(h->heap); //힙 배열 메모리 해제
    free(h); //힙 메모리 해제
}

```

25. Delete\_Heap함수는 동적할당한 부분을 모두 메모리 해제해주는 함수이다.

가장 먼저 heap의 이름을 삭제한다. 0번 인덱스에는 아무 값이 없고 1번 인덱스부터 힙의 끝까지 동적할당한 이름이 존재하기 때문에 for문을 돌려 이름 포인터를 메모리 해제한다.

26. 이름포인터를 모두 해제했으면 Element형으로 동적할당한 힙 배열을 메모리 해제하고 마지막으로 힙타입인 힙을 메모리 해제한다.



## 2.4 실행 창

- 원래의 데이터에 장범준 이름을 추가하여 가나다순으로 삭제가 되는지 확인

```
>> 손님(손흥민) 입장
< 히프 출력 >
1: 손흥민 =>

>> 손님(기성용) 입장
< 히프 출력 >
1: 기성용 => 2: 손흥민 =>

>> 손님(조현우) 입장
< 히프 출력 >
1: 기성용 => 2: 손흥민 => 3: 조현우 =>

>> 손님(기성용) 퇴장
< 히프 출력 >
1: 손흥민 => 2: 조현우 =>

>> 손님(손흥민) 퇴장
< 히프 출력 >
1: 조현우 =>

>> 손님(구자철) 입장
< 히프 출력 >
1: 구자철 => 2: 조현우 =>

>> 손님(이청용) 입장
< 히프 출력 >
1: 구자철 => 2: 조현우 => 3: 이청용 =>

>> 손님(구자철) 퇴장
< 히프 출력 >
1: 이청용 => 2: 조현우 =>

>> 손님(이동국) 입장
< 히프 출력 >
1: 이동국 => 2: 조현우 => 3: 이청용 =>

>> 손님(이동국) 퇴장
< 히프 출력 >
1: 이청용 => 2: 조현우 =>

>> 손님(박주호) 입장
< 히프 출력 >
1: 박주호 => 2: 조현우 => 3: 이청용 =>

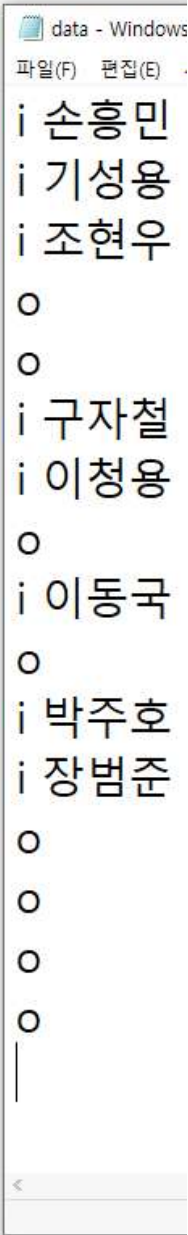
>> 손님(장범준) 입장
< 히프 출력 >
1: 박주호 => 2: 장범준 => 3: 이청용 => 4: 조현우 =>

>> 손님(박주호) 퇴장
< 히프 출력 >
1: 이청용 => 2: 장범준 => 3: 조현우 =>

>> 손님(이청용) 퇴장
< 히프 출력 >
1: 장범준 => 2: 조현우 =>

>> 손님(장범준) 퇴장
< 히프 출력 >
1: 조현우 =>

>> 손님(조현우) 퇴장
< 히프 출력 >
```



C:\Users\dnk46\OneDrive\바탕 화면\2학기 수업자료\자료구조2\실습\week6\_2\실행창  
인해 종료되었습니다.

## 2.5 느낀 점

이번 문제는 l 또는 o로 삽입 또는 삭제 명령을 입력받고 이를 히프를 이용하여 구현하는 문제였습니다. 1번문제와 달리 루트 노드에는 가나다순으로 가장 빠른 이름이 저장되게 해야하기 때문에 최소 히프로 구현하여 프로그래밍을 해보았습니다. 그리고 문자열을 비교해야하기 때문에 strcmp를 사용하여 비교할 수 있도록 했습니다.

처음 프로그래밍을 했을 때, 히프를 배열로 선언하여 작성했지만 메모리의 크기를 아끼기위해 포인터로 선언하여 동적할당으로 해보았습니다. 히프를 동적할당 할 때는 비트시프트 연산을 통해 높이에 맞게 동적할당하여 메모리의 크기를 아낄 수 있었습니다.

이번 문제는 삽입뿐만 아니라 삭제도 구현해야 하는 문제였습니다. 삭제 연산은 루트노드에 있는 가장 빠른 문자열을 반환해야하므로, 책에 있는 코드를 응용하여 삭제 연산을 프로그래밍할 수 있었습니다.

히프의 삭제 연산을 처음 프로그래밍 해보았기 때문에 처음에는 어려운 부분이 있었지만 이번에 레포트를 작성하며 층을 하나씩 내려가는 연산에 대해 이해를 할 수 있었습니다.

이번 과제를 통해 히프의 삽입과 삭제 연산의 알고리즘에 대해 정확히 알 수 있는 기회가 되었고 알고리즘 공부를 더 열심히 해서 앞으로 히프에 관련된 문제가 나온다면 쉽게 풀 수 있도록 노력하겠습니다.



### 3. 우선순위 큐를 하며 느낀 점

이번 우선순위 큐 2문제는 각각 최대힙과 최소힙을 이용하여 구현할 수 있었습니다. 이번 과제를 통해 두 힙의 차이점을 정확히 알 수 있었고 두 힙을 구현할 때 삽입과 삭제연산에 대해서도 알 수 있는 기회가 되었습니다.

전까지는 링크 포인터를 이용한 리스트나 트리를 이용하여 프로그래밍 해왔고, 힙을 처음 이용하여 구현했기 때문에 배열로 구현하는 것과 반복을 이용하여 삽입을 하는 등 모든 것이 생소하게 느껴졌던 것 같습니다. 하지만 이번 과제를 계기로 힙의 알고리즘에 대해 이해할 수 있었고 이런 방법으로도 구현할 수 있구나 하는 것을 느낄 수 있었습니다.

이번에 힙을 프로그래밍해본 경험으로 앞으로 힙에 대한 문제가 나온다면 틀리지 않고 자신있게 풀 수 있도록 하겠습니다. 힙을 이용한 머신 스케줄링이나 허프만 코드에 대한 문제를 풀어보지는 않았지만 시험공부를 하며 힙을 이용하여 이를 직접 프로그래밍 해볼 것이고 공부해볼 것입니다.