2021/10/09

자료구조2 실습 레포트

과목명 | 자료구조2 실습

담당교수 | 홍 민 교수님

학과 | 컴퓨터소프트웨어공학과

학년 | 2학년

학번 | 20204059

이름 | 이예빈

목 차

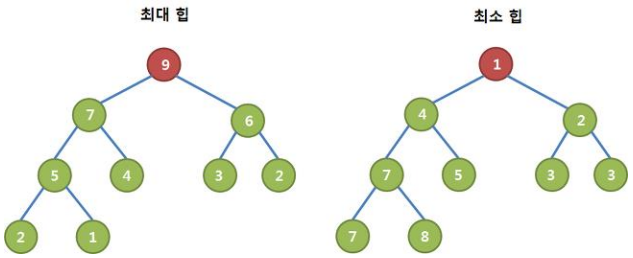
1. 우선 순위 큐로 구현하는 동물 히프

1.1 분석

1.1.1 히프의 개념

우선순위 큐(priority queue)는 각 요소들이 우선 순위값을 가지고 있으며, 우선 순위가 높은 데이터가 먼저 나가게 되는 구조이다. 우선순위 큐에는 가장 우선 순위가 낮은 요소를 먼저 삭제하는 최소 우선순위 큐, 그리고 반대로 우선 순위가 높은 요소가 먼저 삭제되는 최대 우선순위 큐가 있다.

히프(heap)는 우선순위 큐를 구현하는 가장 효율적인 구조이다. 히프는 완전 이진 트리의 일종으로, 여러 개의 값들 중에서 가장 큰 값이나 가장 작은 값을 빠르게 찾아내는 경우에 매우 유리한 자료 구조이다. 부모 노드의 키 값이 자식 노드의 키 값보다 항상 큰 이진 트리로 구현된 것을 최대 히프(max heap),0 반대로 노드의 키 값이 자식 노드의 것보다 항상 작은 히프를 최소 히프(min heap)라고 한다.

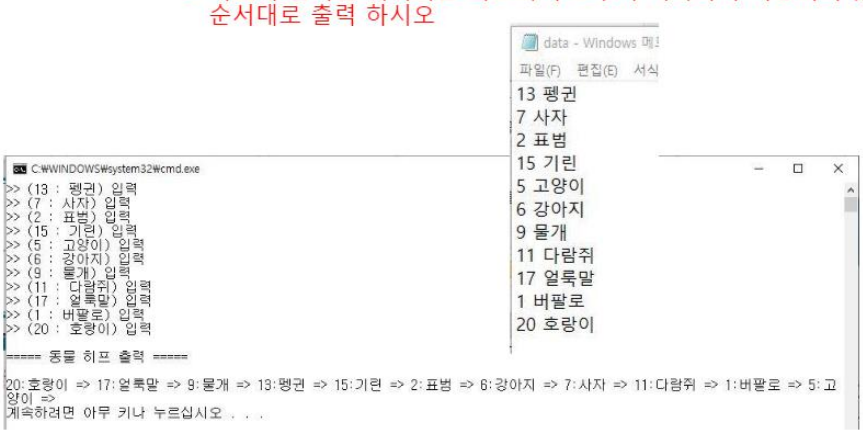


히프는 완전 이진 트리이므로 루트 노드부터 시작하여, 각 레벨의 왼쪽부터 순회한다고 하였을 때 각각의 노드에 차례대로 번호를 붙일 수 있다. 1번부터 시작하는 번호들을 배열의 인덱스로 생각하여 배열에 히프의 노드들을 저장할 수 있다. 따라서, 부모 노드와 자식 노드간의 관계를 배열의 인덱스로 쉽게 나타낼 수 있는데, 왼쪽 자식에 인덱스는 부모 노드의 인덱스에 2를 곱한 값이고, 오른쪽 자식의 인덱스는 부모 노드 인덱스에 2를 곱하여 1을 더한 값이다.

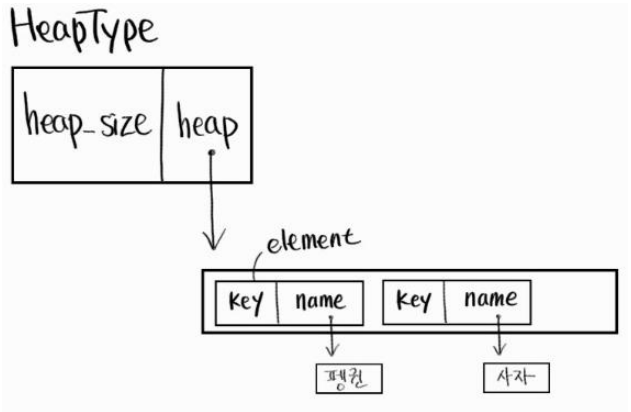
히프의 연산에서 중요한 삽입 연산과 삭제 연산은 다음 [1.1.2 문제 분석]을 통해 문제를 분석하고 문제의 예시를 바탕으로 그 연산 동작 과정을 자세 실펴보도록 하자.

1.1.2 문제 분석

- 동물 히프
  - 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 출력 하시오



파일에 우선 순위와 동물의 이름이 각 줄에 저장되어 있다. 따라서 히프의 구조체는 아래와 같이 구성할 수 있다. 기존에는 히프의 배열 길이를 기호 상수로 지정하여 정적으로 할당하였지만, 이 프로그램에서는 파일의 데이터를 먼저 읽어 그 크기 만큼 배열을 동적 할당 하도록 하자. 동물의 이름 또한 문자열의 길이를 구하여 동적 할당하도록 하자.



HeapType의 구조체는 히프에 삽입되어 있는 요소의 개수를 의미하는 heap_size와 동적 할당 받게 되는 배열의 주소를 가리키는 포인터 heap을 멤버로 갖는다. element형 배열은 정수형 key 값과 문자 배열을 가리키는 포인터 name을 멤버로 갖는다.

삽입 연산

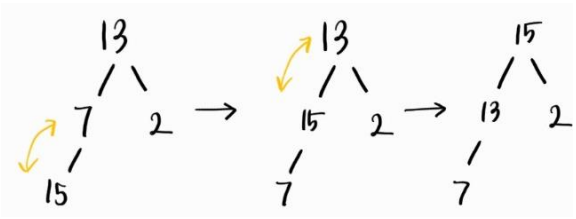
예시 실행창에서 "20:호랑이"부터 히프가 출력되는 것을 보아 최대 히프(max heap)라는 것을 알 수 있다. 최대 히프 삽입 연산은 가장 마지막 위치에 새로운 키 값을 가진 노드를 삽입하는 것으로 가정하고, 부모 노드와 키 값을 비교하여 더 큰 경우 두 노드의 데이터를 교환하기를 반복한다. 부모 노드는 자식 노드로 이동을 하게 되며, 마지막에 구해지는 위치의 인덱스 값에 새로운 데이터를 삽입하게 된다.

이 문제의 예시 실행창대로, 13(펭귄)부터 20(호랑이)까지 삽입되는 과정을 이진 탐색 트리로 나타내면 아래와 같다.

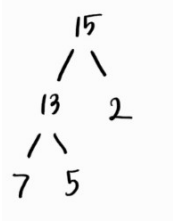
13(펭귄), 7(사자), 2(표범) 삽입

위의 세 데이터를 삽입할 때에는 부모 노드인 13 이 가장 크기 때문에 데이터의 교환이 발생하지 않는다. 아래 15(기린) 데이터를 삽입하는 경우에는, 차례대로 부모노드 7, 그리고 13 과의 교환이 이루어진다.

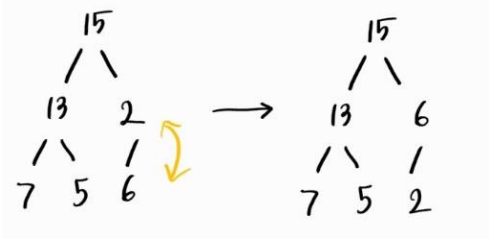
15(기린) 삽입

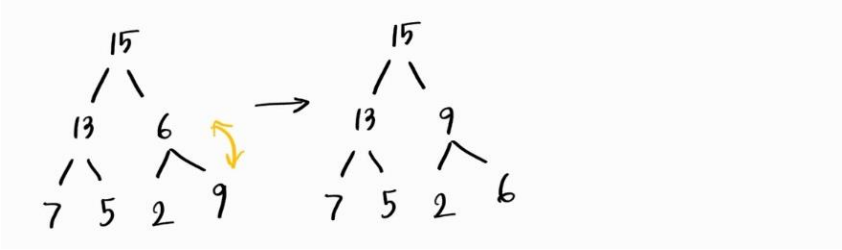


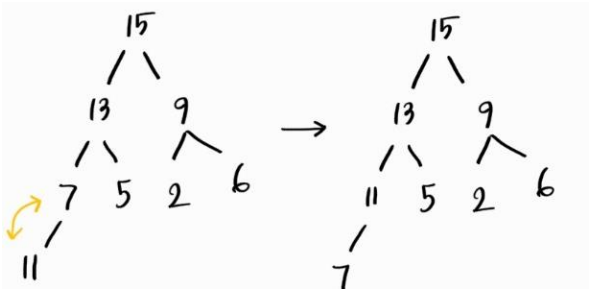
5(고양이) 삽입

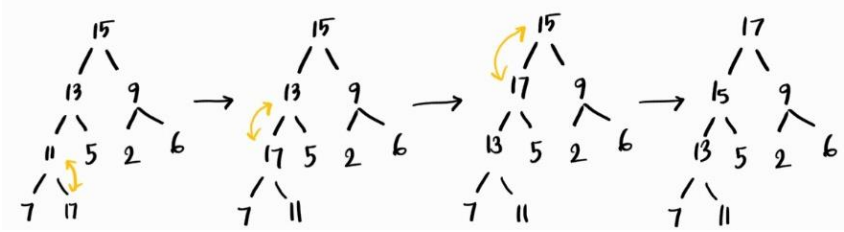


6(강아지) 삽입



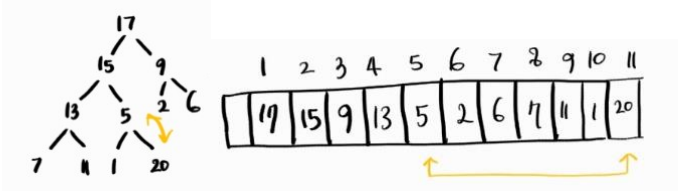
9(물개) 삽입 

11(다람쥐) 삽입 

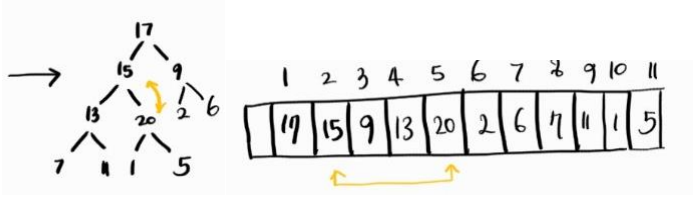
17(얼룩말) 삽입 

11(다람쥐), 1(버팔로) 삽입 

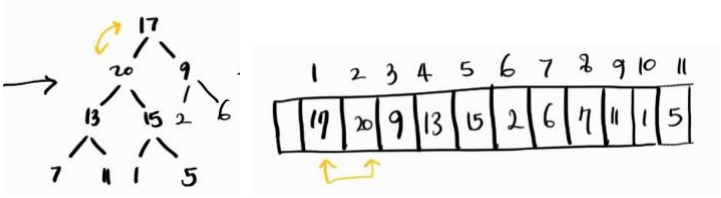
마지막으로 20(호랑이)를 삽입하는 과정을 배열로도 살펴보자.

20(호랑이) 삽입 

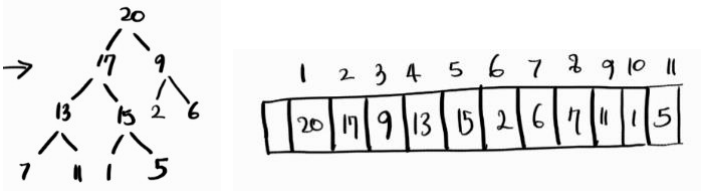
20 호랑이는 가장 마지막 위치인 11 번 인덱스에 우선 삽입된다. 이후 부모 노드인 5 번 인덱스인 5 와 비교하여 두 값을 교환하도록 한다.



이번에는 부모 노드인 2 번 인덱스 15 와 교환하도록 한다.



1 번 인덱스의 17 과도 교환이 이루어진다.

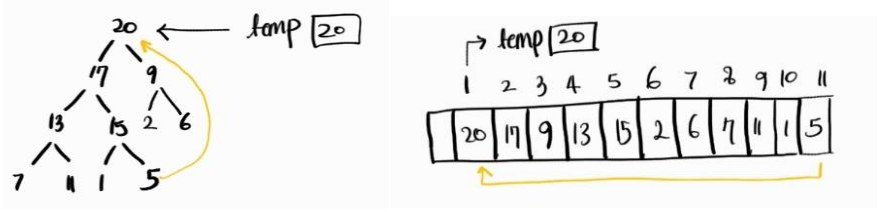


결국 가장 큰 값인 20 이 트리의 루트 노드(1 번 인덱스)에 도달하게 되어 모든 데이터들이 삽입된 후 최대 히프가 완성된다.

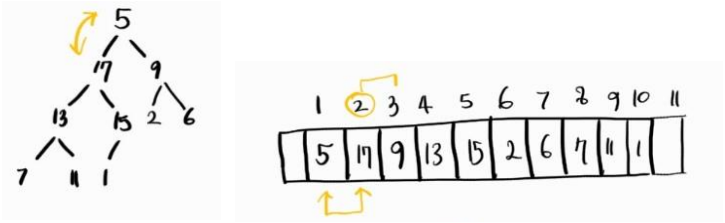
삭제

삭제 연산은 우선 순위가 가장 높은 루트 노드(1 번 인덱스의 요소)의 값이 삭제되어 반환된다. 루트 노드의 값이 삭제가 되고 나서는, 다시 그다음으로 우선순위가 높은 값을 루트 노드로 옮겨주어야 하고, 히프 구조에 부합하도록 나머지 요소들도 적절히 이동을 시켜야 한다. 반환할 값은 임시로 변수에 저장해 놓고, 가장 마지막 노드의 값 또한 임시로 변수에 저장해 놓는다. 그리고 루트 노드의 자리에 이동시켰다고 가정하며, 두 자식 노드 중 더 큰 값과 교환하기를 반복한다. 이 반복은 가장 마지막 노드였던 값과 동일하거나 더 작은 값이 나오기 전까지 반복하도록 한다.

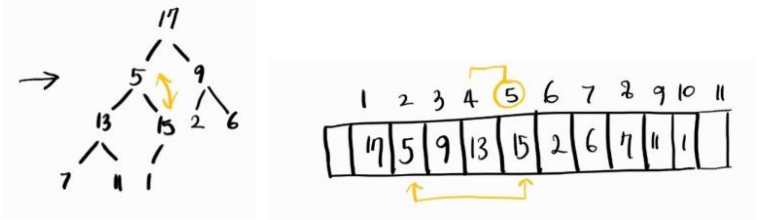
위에서 모든 요소의 삽입이 완료된 히프에서 한 번의 삭제가 이루어지는 연산을 이진 트리, 그리고 배열의 구조로 표현된 다음 그림들을 통해 살펴보자.



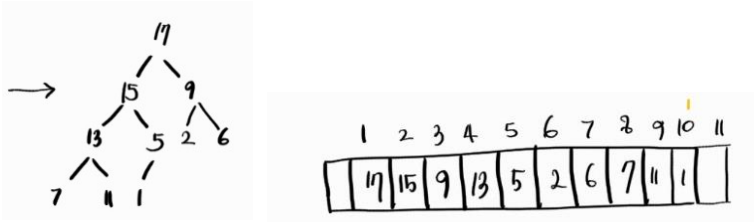
위의 히프에서 루트 노드에 존재하는 20 의 값이 삭제된다. 20 을 temp 에 임시로 저장해놓고, 가장 마지막 노드의 값이었던 5 를 루트 노드로 이동시킨다.



자식 노드들 중 왼쪽 자식 노드 17 이 오른쪽 자식 노드 9 보다 크므로, 5 를 17 과 교환하도록 하자.



이번에는 오른쪽 자식 노드의 값이 15 로 더 크기 때문에 5 와 15 를 교환하도록 한다.



자식 노드인 1 이 5 보다 작기 때문에 더 이상의 교환이 이루어지지 않는다. 이로써 모든 삭제 연산이 끝났다. 마지막으로 temp 에 담아 놓았던 20 의 값을 반환한다.

1.2 소스 코드

```c
/*
	작성자: 이예빈(20204059)
	작성일: 2021. 10. 07
	프로그램명: 우선순위와 동물들의 이름을 저장하여 히프에 추가하는 프로그램 (최대 히프)
*/

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

typedef struct element {
	int key;
	char* name;
}element;

typedef struct { // 히프 구조체
	element* heap;
	int heap_size;
}HeapType;

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

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

void init(HeapType* h) {
	h->heap_size = 0;
}

 // 최대 히프 구현하는 삽입 함수
void insert_max_heap(HeapType* h, element item) {
	int i;
	i = ++(h->heap_size);

	while ((i != 1) && (item.key > h->heap[i / 2].key)) {
		h->heap[i] = h->heap[i / 2];
		i /= 2;
	}
	h->heap[i] = item;
}
```

```c
int main() {
	FILE* fp;
	element tmp;		//임시 구조체
	char name[20];		//임시로 저장할 이름 문자 배열
	HeapType* heap;		//히프 구조의 포인터
	int i=0, size=0;//반복문 제어 변수, 히프 전체 출력을 위한 배열 크기

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

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

	while (!feof(fp)) {
		fscanf(fp, "%d %s", &tmp.key, name);
		i++;
	}
	rewind(fp);
	heap->heap = (element*)malloc(sizeof(element) * (i+1));

	while (!feof(fp)) {
		fscanf(fp, "%d %s", &tmp.key, name);
		tmp.name = (char*)malloc(strlen(name) + 1);//문자열 포인터에 메모리 할당
		if (tmp.name == NULL)error("문자 배열 동적 할당 실패");
		else {
			strcpy(tmp.name, name);		//문자열 복사
			printf(">> (%d : %s) 입력\n", tmp.key, tmp.name);
			insert_max_heap(heap, tmp);	//최대 히프에 삽입
		}
	}
	size = heap->heap_size;
	printf("\n\n===== 동물 히프 출력 =====\n\n");
	for (i = 1; i <= size; i++) {	//배열의 순서대로 출력
		printf("%d:%s =>", heap->heap[i].key, heap->heap[i].name);
		free(heap->heap[i].name);
	}
	printf("\n\n");

	free(heap->heap);	//히프 구조체 메모리 해제
	free(heap);		//히프의 모든 메모리 해제
	fclose(fp);		//파일 닫기
	return 0;
}
```

1.3 소스 코드 분석

```
/*
    작성자: 이예빈(20204059)
    작성일: 2021. 10. 07
    프로그램명: 우선순위와 동물들의 이름을 저장하여 히프에 추가하는 프로그램 (최대 히프)
*/

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

1. 주석에 작성자, 작성일, 프로그램명을 작성한다. 이후 필요한 헤더 파일들을 추가한다. 동적 할당에 필요한 표준 라이브러리 헤더, 그리고 문자열 처리 라이브러리 함수를 추가한다.

```
typedef struct element {
    int key;
    char* name;
}element;

typedef struct { // 히프 구조체
    element* heap;
    int heap_size;
}HeapType;
```

2. 프로그램에 필요한 구조체들을 정의한다. 정수형 키 key, 이름을 저장할 문자형 포인터 name 을 멤버로 하는 element 구조체를 정의한다. 그리고 히츠 구조체는 히프의 요소들을 가리키는 포인터 heap, 그리고 히프의 크기 hea_size 를 멤버로 하는 히프 구조체 HeapType 을 선언한다.

main 함수의 동작을 살펴 보기 전, 프로그램에서 사용되는 함수들의 설명을 먼저 짚고 넘어가자.

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

3. 동적 할당 오류 또는 파일 입출력 오류가 발생하는 경우, message 문자열을 출력하고 프로그램 실행을 종료하는 error 함수이다.

```
HeapType* create() {
    return (HeapType*)malloc(sizeof(HeapType));
}

void init(HeapType* h) {
    h->heap_size = 0;
}
```

4. 히프 구조체를 동적 할당하하여 반환하는 create 함수, 그리고 히프의 heap_size 를 0 으로 초기화하는 init 함수이다.

```
// 최대 히프 구현하는 삽입 함수
void insert_max_heap(HeapType* h, element item) {
    int i;          //히프 배열 인덱스 값
    i = ++(h->heap_size);  //히프의 크기를 1 증가시킨 값을 i에 저장

    //첫번째 인덱스값이 아니고, item의 키 값이 부모 노드의 키 값보다 큰 경우
    while ((i != 1) && (item.key > h->heap[i / 2].key)) {
        h->heap[i] = h->heap[i / 2];    //i번 인덱스의 요소(자식 노드)에 i/2번 인덱스의 요소(부모 노드) 대입
        i /= 2;                         //i/2번 인덱스로 이동 (부모 노드)
    }
    h->heap[i] = item; //i번 인덱스에 새로운 item 대입
}
```

5. 가장 큰 값이 루트 노드가 되는 최대 히프를 구현하기 위해 새로운 삽입 함수 insert_max_heap 함수이다. 히프 구조체의 포인터와 새로 삽입할 element 요소의 item 변수를 매개변수로 전달받는다.

우선 히프 배열의 인덱스 값을 나타내는 정수 변수 i 를 선언하고, 새로운 노드가 삽입되어야 하므로 히프의 heap_size 를 1 증가시키고, i 에 해당 값을 대입한다.

6. 1 번 인덱스 값(첫번째 루트 요소)가 아니고, 부모 노드의 키 값이 부모 노드의 키 값보다 큰 경우에는 값의 교환이 이루어져야 한다. 따라서 i 번 노드에 i/2 번 노드인 부모의 노드가 대입된다. (부모 노드가 아래 자식 노드로 이동) 그리고 다시 i/2 번 인덱스로 이동하게 된다. (위로 이동하여 비교)

7. 마지막으로 모든 반복문이 끝나고, 최대 히프를 만족시키도록 하는 i 번 노드에 item 을 집어넣는다.

이제 다음으로, main 함수의 동작을 살펴보자.

```
int main() {
    FILE* fp;
    element tmp;        //임시 구조체
    char name[20];      //임시로 저장할 이름 문자 배열
    HeapType* heap;     //히프 구조의 포인터
    int i = 0, size = 0;//반복문 제어 변수, 히프 전체 출력을 위한 배열 크기
```

8. 파일 포인터 fp, element 형 임시 구조체 tmp, 임시로 이름을 저장할 문자 배열 name, 그리고 히프 구조 포인터 heap 와 정수형 반복문 제어 변수 i 와 히프 배열 크기를 나타낼 size 값을 0 으로 초기화한다.

```
    heap = create();    //히프 생성
    init(heap);         //히프 초기화

    fp = fopen("data.txt", "rt");
    if (fp == NULL) {
        fprintf(stderr, "파일 열기 오류\n");
        exit(1);
    }
```

9. create 함수를 호출하여 히프 구조체를 동적 할당 받고 반환값을 heap 포인터에 저장한다. init 함수에 heap 를 인수로 전달하여 히프를 초기화한다. 히프의 size 멤버를 0 으로 초기화하도록 한다.

10. data.txt 파일을 읽기 모드로 연다.

```
    while (!feof(fp)) {
        fscanf(fp, "%d %s", &tmp.key, name);
        i++;
    }
    rewind(fp);
    heap->heap = (element*)malloc(sizeof(element) * (i+1));
```

11. 우선적으로 파일의 데이터 개수를 알아내기 위해, 정수와 문자열을 읽어 임시로 tmp 의 key 멤버와 name 문자열에 저장하고 0 으로 초기화했던 i 의 값을 증가시킨다.

12. 이후 rewind 함수를 이용해 다시 파일 포인터를 파일의 처음으로 이동시키고, heap 구조체의 heap 포인터에 (i+1)개의 element 요소를 가진 배열을 동적 할당한다. 히프에는 1 번 인덱스부터 요소가 저장되므로, 읽은 데이터의 개수보다 값이 하나 더 큰 배열을 할당하도록 한다.

```
while (!feof(fp)) {
    fscanf(fp, "%d %s", &tmp.key, name);
    tmp.name = (char*)malloc(strlen(name) + 1);//문자열 포인터에 메모리 할당
    if (tmp.name == NULL)error("문자 배열 동적 할당 실패");
    strcpy(tmp.name, name);                    //문자열 복사
    printf(">> (%d : %s) 입력 \n", tmp.key, tmp.name);
    insert_max_heap(heap, tmp);    //최대 히프에 삽입
}
```

13. 다시 파일을 읽어 히프에 각 요소를 삽입하도록 한다. tmp 구조체의 key 멤버, 그리고 name 문자열에 데이터를 입력 받는다. tmp 구조체의 name 포인터에 (읽은 name 문자열의 길이+1) 만큼 문자 배열을 동적 할당 한다. name 문자 배열을 할당한 문자 배열 공간에 복사한다. 입력한 데이터를 출력하고, insert_max_heap 에 히프의 주소와 구조체 tmp 를 전달하여 최대 히프에 삽입한다.

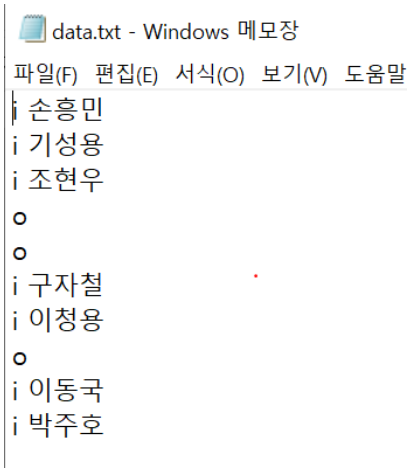
```
size = heap->heap_size;
printf("\n\n===== 동물 히프 출력 ===== \n\n");
for (i = 1; i <= size; i++) {  //배열의 순서대로 출력
    printf("%d:%s => ", heap->heap[i].key, heap->heap[i].name);
    free(heap->heap[i].name);
}
printf("\n\n");
```

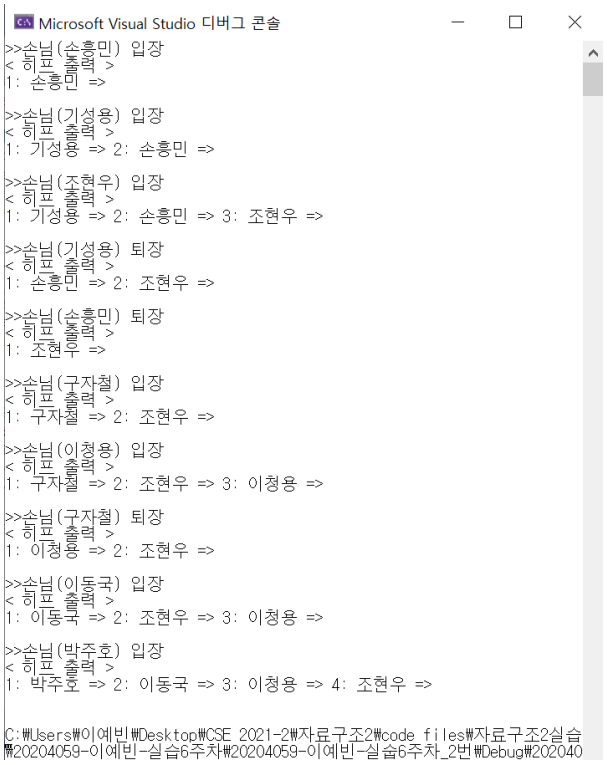
14. size 변수에 현재 히프에 삽입되어 있는 요소의 개수인 heap_size 멤버를 대입한다. 이후 1~size 인덱스까지 히프 배열에 저장되어 있는 요소들을 양식에 맞게 출력하도록 한다. 메모리 해제를 위해서, 각 요소를 출력한 후에 할당했던 문자 배열은 메모리를 해제하도록 한다.

```
free(heap->heap);  //히프 구조체 메모리 해제
free(heap);        //히프의 모든 메모리 해제
fclose(fp);        //파일 닫기
return 0;
}
```

15. 마지막으로 히프의 heap 포인터에 할당된 배열을 모두 메모리 해제하고, heap 구조체의 메모리 또한 메모리 해제한다. 파일을 닫고 0 을 리턴하여 main 함수를 종료한다.

1.4 실행창


data.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말
i 손흥민
i 기성용
i 조현우
o
o
i 구자철
i 이청용
o
i 이동국
i 박주호

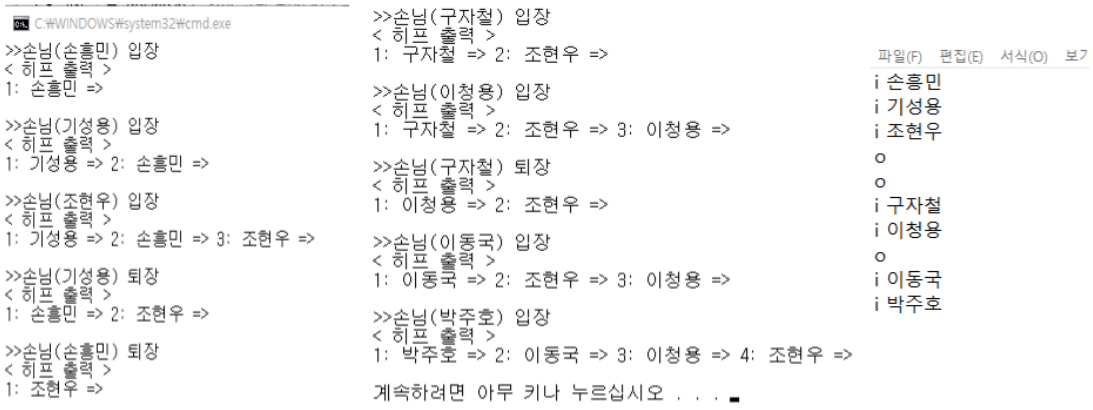

Microsoft Visual Studio 디버그 콘솔
>>손님(손흥민) 입장
< 히프 출력 >
1: 손흥민 =>
>>손님(기성용) 입장
< 히프 출력 >
1: 기성용 => 2: 손흥민 =>
>>손님(조현우) 입장
< 히프 출력 >
1: 기성용 => 2: 손흥민 => 3: 조현우 =>
>>손님(기성용) 퇴장
< 히프 출력 >
1: 손흥민 => 2: 조현우 =>
>>손님(손흥민) 퇴장
< 히프 출력 >
1: 조현우 =>
>>손님(구자철) 입장
< 히프 출력 >
1: 구자철 => 2: 조현우 =>
>>손님(이청용) 입장
< 히프 출력 >
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(박주호) 입장
< 히프 출력 >
1: 박주호 => 2: 이동국 => 3: 이청용 => 4: 조현우 =>
C:₩Users₩이예빈₩Desktop₩CSE 2021-2₩자료구조2₩code files₩자료구조2실습
₩20204059-이예빈-실습6주차₩20204059-이예빈-실습6주차_2번₩Debug₩202040

1.5 느낀점

1번 문제를 최대값 혹은 최소값이 우선적으로 삭제되는 '우선순위 큐'를 배열의 구조로 구현하는 히프의 개념과 구조, 그리고 기본적인 연산에 대해 깊이 공부할 수 있었다. 특히나 이진 트리를 공부한 후에, 히프에 대해 배워 보니, 사실상 배열인 구조를 이진 트리로 시각화 할 수 있다는 점이 매우 새로웠다. 특히나 삽입과 삭제 연산의 알고리즘이 매우 독특했다. 가장 마지막 노드에 우선적으로 위치해 있다가, 부모 노드의 우선 순위를 비교하여 조건이 만족되면 부모 노드를 강등시키기를 반복하며, 삽입될 위치를 탐색하는 방법이 매우 참신하게 느껴졌다. 이렇게 우선 순위라는 큐를 통해 우선순위에 따라 특정한 값들을 반환해야 하는 경우가 어떤 것이 있을지 앞으로 더 알아가고 싶다.

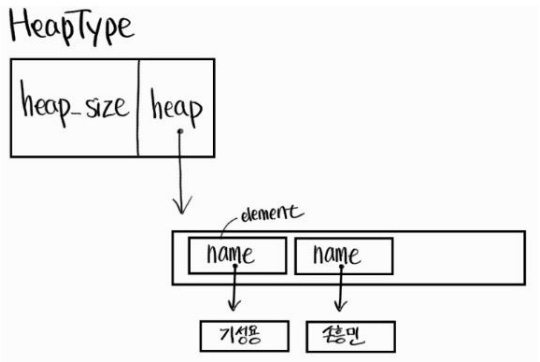
2. 우선 순위 큐로 구현하는 손님 관리 프로그램

- 손님 관리
  - data.txt에 손님의 입장과 퇴장이 이름과 함께 입력되어있다. 각 입장과 퇴장을 히프에 적용하고 이를 히프에 저장된 순서대로 출력하시오.
    - i는 입장, o은 퇴장을 뜻 함
    - 우선 순위는 앞 사람이 퇴장하면 이름순(가나다순)으로 적용됨



2.1 문제 분석

이 문제는 각 노드가 문자열을 키 값으로 하며, 루트 노드(1번 인덱스)가 가장 작은 키 값(사전순으로 맨 앞에 위치하게 되는 문자열)을 가지는 노드인 최소 히프를 구현하는 프로그램이다. 파일에는 입장과 퇴장을 알리는 명령어 문자('i' 또는 'o')와 손님의 이름이 저장되어 있다. 이 파일을 읽어 최소 히프에 삽입, 그리고 최소 히프로부터 삭제를 하여 출력을 하는 프로그램을 구현해야 한다.



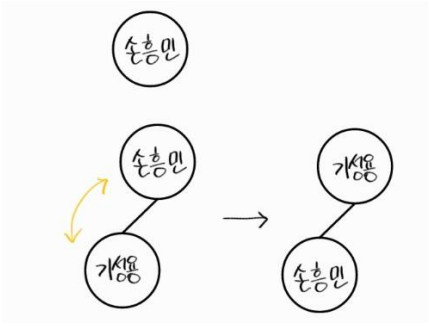
이 문제의 히프 구조체는 위와 같이 그림으로 나타낼 수 있다. 히프 구조체 HeapType은 히프에 삽입되어 있는 요소들의 개수인 heap_size와 히프 배열을 가리키는 heap 포인터를 멤버로 갖는

다. 히프 배열은 name 문자열 포인터를 멤버로 갖는 element 요소들로 구성되어 있다.

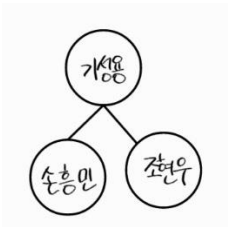
[1번 문제의 문제 분석]에서 설명한 히프의 삽입, 삭제 연산과 방식이 동일하다. 그러나 2번 문제에서는 노드의 키 값이 문자열이라는 점, 그리고 루트 노드에 사전 순으로 맨 앞에 있는 문자열이 위치하는 최소 히프라는 점이 다르다. 삽입의 경우에는, 새로운 데이터를 마지막 노드로 추가한 후에 부모 노드와 문자열을 비교하기를 반복하여 최종 삽입의 위치를 결정하게 된다. 삭제의 경우, 루트 노드의 값을 임시로 저장해 놓은 후, 마지막 노드의 값을 루트 노드부터 시작하여 차례대로 더 작은 문자열을 가진 자식 노드 문자열을 비교하여 교환하기를 반복하고 마지막에 임시로 저장해 놓은 기존 루트 노드의 값을 반환한다.

다음 그림의 예시들을 통해 문자열 키 값을 이용한 삽입과 삭제를 간단히 살펴보자.

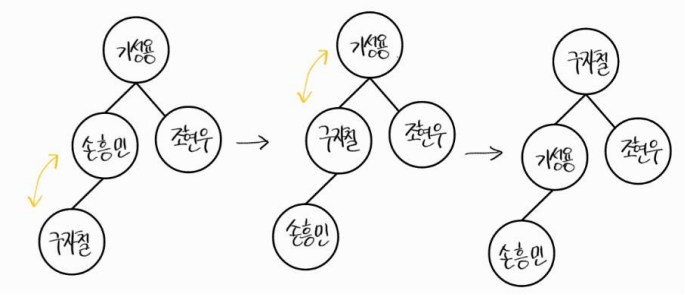
삽입



'손흥민'이 입장하고, 뒤이어 '기성용'이 입장한다. '기성용'이 입장했을 때, 부모 노드인 '손흥민'과 문자열을 비교한다. '기성용'의 문자열이 사전 순으로 '손흥민'보다 앞에 위치하므로, 두 노드를 교환한다.



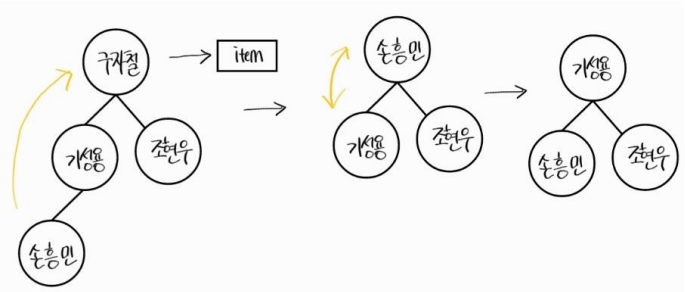
'조현우'가 입장한다. '기성용'과 문자열을 비교하여 사전 순으로 뒤에 위치하므로 값의 교환이 이루어지지 않는다.



'구자철'이 입장한다. '구자철'의 부모 노드의 '손흥민'과 비교하여 값을 교환한다. '구자철'은 다시 부모 노드인 '기성용'과 비교하여 값을 교환한다. 결국 '구자철'은 사전 순으로 맨 앞에 위치하는 단어이므로, 결국 루트 노드에 위치하게 된다.

삭제

이번에는 완성된 히프로부터 한 번의 삭제가 이루어지는 연산을 그림으로 살펴보자.



루트 노드의 '구자철'이 반환되고 삭제되는 값이다. 이를 item에 저장해 놓는다. 가장 마지막 노드의 '손흥민'이 루트 노드의 위치에서부터, 사전 순으로 더 앞에 위치한 자식 노드 중 하나와 비교하기를 반복한다. 루트에 위치한 '손흥민'은 자식 노드들 중 사전 순으로 더 앞에 위치한 왼쪽 자식 노드인 '기성용'과 값을 교환하게 된다.

item에 들어있던 '구자철'은 반환되어 완전히 삭제가 되고, 교환 과정을 통해서 다시 새로운 최소 히프가 완성된다. 사전 순으로 맨 앞에 위치한 '기성용'이 루트 노드의 값이 되어 최소 히프가 완성된 것이다.

이 문제에서 또 고려해야 할 점은 히프의 크기(배열의 길이)를 히프 완전 이진 트리의 마지막 레벨의 노느들을 모두 고려하여 결정해야 한다는 점이다. 또한, 노드의 키 값이 문자열이기 때문에 문자열을 동적 할당해야 한다는 점, 그리고 문자열 처리 함수를 사용해서 두 노드의 키 값을 비교해야 한다는 점까지 고려해야 한다. 더 자세한 설명은 [소스 코드 분석] 단계에서 그 과정을 자세히 살펴보도록 하자.

2.2 소스 코드

```c
/*
    작성자 : 이예빈(20204059)
    작성일 : 2021.10.07
    프로그램명 : 최소 히프 구조의 우선순위 큐를 이용한 손님 관리 프로그램
*/

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

typedef struct element {
    char* name;
}element;

typedef struct {   // 히프 구조체
    element* heap;
    int heap_size;
}HeapType;

void error(char* message) {
    fprintf(stderr, "%s\n", message);
    exit(1);
}
HeapType* create() {
    return (HeapType*)malloc(sizeof(HeapType));
}

void init(HeapType* h) {
    h->heap_size = 0;
}

int compare(char* e1, char* e2) {
    return strcmp(e1, e2);
}

// 사전 순으로 맨 앞에 있는 단어가 루트 노드가 되는 최소 히프
void insert_min_heap(HeapType* h, element item) {
    int i;
    i = ++(h->heap_size);

    // 1번 요소가 아니고, 삽입하려는 요소의 문자열이 i/2번(부모) 인덱스의 문자열보다 사전 순으로 앞에 있는 경우
    while ((i != 1) && (compare(item.name, h->heap[i / 2].name) < 0)) {
        h->heap[i] = h->heap[i / 2];   // i번 인덱스(자식노드)에 i/2번 인덱스(부모노드) 대입
        i /= 2;//부모 노드를 따라 올라감
    }
    h->heap[i] = item; // 최종 i번 인덱스 위치에 item 삽입
}

// 최소 히프의 루트 노드부터 삭제
element delete_min_heap(HeapType* h) {
    int parent, child; // 부모/자식 노드 인덱스
    element item, temp;//반환할 item, 임시 temp

    item = h->heap[1]; // 반환할 요소는 1번 인덱스의 요소
    temp = h->heap[(h->heap_size)--];  // 마지막 요소를 temp에 저장하고, 힙 크기 1 감소
    parent = 1, child = 2; // 부모, 자식 노드 인덱스는 1,2부터 시작

    // child가 힙의 마지막 요소에 도달할 때까지 반복
    while (child <= h->heap_size) {
        // 힙의 마지막 요소가 아니고, child(왼쪽 자식노드)가 child+1(오른쪽 자식 노드)보다 문자열이 뒤에 있는 것이라면
        if ((child < h->heap_size) && (compare(h->heap[child].name, h->heap[child + 1].name) > 0))
            child++;   // 1증가하여 오른쪽 자식 노드와 비교하도록

        // temp의 문자열이 child 인덱스의 문자열보다 사전 순으로 앞에 있거나 같다면 반복문 종료
        if (compare(temp.name, h->heap[child].name) <= 0)
            break;

```

```
68      h->heap[parent] = h->heap[child];
69      parent = child;//부모의 인덱스는 자식의 인덱스로 이동
70      child *= 2;      //왼쪽 자식 노드 인덱스로 이동
71    }
72    h->heap[parent] = temp;//최종 부모 인덱스에 마지막 요소 값을 대입
73    return item;
74  }
75
76  //순서대로 히프의 요소 출력
77  void display_heap(HeapType* h) {
78    int i, count = h->heap_size;
79    printf("< 히프 출력 >\n");
80    for (i = 1; i <= count; i++) {
81      printf("%d: %s =>", i, h->heap[i].name);
82    }
83    printf("\n\n");
84  }
85
86  int main() {
87    FILE* fp;
88    element tmp;      //임시 구조체
89    char command, name[20];//파일의 명령어 문자, 임시 문자 배열
90    int i = 0, size = 0, max_count = 0, heap_size = 0; //히프의 배열 크기 구하기
91    HeapType* heap;
92
93    heap = create();   //히프 구조체 동적 할당
94    init(heap);      //히프 초기화
95
96    fp = fopen("data.txt", "rt");
97    if (fp == NULL)error("파일 열기 실패");
98
99    //파일 읽어 히프 노드의 개수(배열 크기) 결정
100   while (!feof(fp)) {
101     fscanf(fp, "%c", &command);
102     if (command == 'i') {
103       fscanf(fp, "%s", name);
104       size++;
105     }
106     else if (command == 'o') size--;
107     //input, output 결과에 의한 최대 수용 가능한 배열의 길이 max_count 구하기
108     if (size > max_count) max_count = size;
109   }
110   rewind(fp);
111
112   for (i = 1;i <= max_count;i *= 2)
113     heap_size = i;
114   heap_size *= 2;//마지막 레벨에 모든 노드가 채워지도록 히프의 크기 결정
115   printf("heap_size = %d \n", heap_size);
116
117   heap->heap = (element*)malloc(sizeof(element) * (heap_size));//히프의 배열 공간 동적 할당
118
119   //다시 파일을 읽어 읽어 히프 삽입 또는 삭제
120   while (!feof(fp)) {
121     fscanf(fp, "%c", &command);
122     if (command == 'i') {
123       fscanf(fp, "%s", name);
124       printf(">>손님(%s) 입장 \n", name);
125
126       //name 멤버에 문자 배열 동적 할당
127       tmp.name = (char*)malloc(sizeof(char) * (strlen(name) + 1));
128       strcpy(tmp.name, name);
129       insert_min_heap(heap, tmp);//최소 히프에 삽입
130       display_heap(heap);      //히프의 요소를 순서대로 출력
131     }
```

```
132     else if (command == 'o') {
133         tmp = delete_min_heap(heap);    //최소 히프에서 삭제한 값을 tmp에 대입
134         printf(">>손님(%s) 퇴장 \n", tmp.name);
135         free(tmp.name);    //tmp의 name 문자 배열 메모리 해제
136         display_heap(heap);
137     }
138 }
139
140 //남아있는 배열 요소들의 문자 배열 메모리 해제
141 for (i = 1;i <= size;i++)
142     free(heap->heap[i].name);
143 free(heap->heap);  //히프 배열 공간 메모리 해제
144 free(heap);    //히프 구조체 메모리 해제
145
146 fclose(fp);//파일 닫기
147 return 0;
148 }
```

2.3 소스 코드 분석

```
/*
    작성자 : 이예빈(20204059)
    작성일 : 2021.10.07
    프로그램명 : 최소 히프 구조의 우선순위 큐를 이용한 손님 관리 프로그램
*/

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

1. 주석에 작성자, 작성일, 프로그램명을 작성하고, 프로그램에 필요한 헤더 파일들을 추가한다. 문자열 처리 헤더 파일, 그리고 동적 할당에 필요한 표준 라이브러리 헤더 파일을 추가한다.

```
typedef struct element {
    char* name;
}element;

typedef struct {    // 히프 구조체
    element* heap;
    int heap_size;
}HeapType;
```

2. 프로그램에서 사용되는 구조체를 정의한다. 히프의 요소를 나타내는 element 구조체는 name 문자형 포인터를 멤버로 갖는다. 히프 구조체 HeapType 은 element 형 배열을 가리키는 포인터 heap, 그리고 히프에 삽입된 요소의 개수를 저장하는 heap_size 정수형 변수를 멤버로 갖는다.

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

3. 파일 입출력 또는 메모리 동적 할당에서 오류가 발생한 경우 message 문자열을 출력하고 프로그램을 종료하는 error 함수이다.

```
HeapType* create() {
    return (HeapType*)malloc(sizeof(HeapType));
}

void init(HeapType* h) {
    h->heap_size = 0;
}

int compare(char* e1, char* e2) {
    return strcmp(e1, e2);
}
```

4. 히프 구조체를 동적 할당하는 create 함수, 그리고 히프의 heap_size 멤버 변수를 0 으로 초기화하는 init 함수이다.

5. 두 문자열 e1, e2 를 매개변수로 전달받아 두 문자열을 사전 순으로 비교한 값의 결과를 반환하는 compare 함수이다. 첫번째 문자열 e1 이 사전 순으로 앞에 위치하면 음수 값을, 뒤에 위치하면 양수 값을, 두 문자열이 동일한 문자열이면 0 을 반환한다.

아래는 히프 구조체 포인터와 새로운 요소 item 을 매개 변수로 전달받아, 새로운 노드를 사전 순으로 맨 앞에 있는 문자열이 루트 노드가 되는 최소 히프에 삽입하는 insert_min_heap 함수이다.

```
//사전 순으로 맨 앞에 있는 단어가 루트 노드가 되는 최소 히프
void insert_min_heap(HeapType* h, element item) {
    int i;
    i = ++(h->heap_size);

    //1번 요소가 아니고, 삽입하려는 요소의 문자열이 i/2번(부모) 인덱스의 문자열보다 사전 순으로 앞에 있는 경우
    while ((i != 1) && (compare(item.name, h->heap[i / 2].name) < 0)) {
        h->heap[i] = h->heap[i / 2];    //i번 인덱스(자식노드)에 i/2번 인덱스(부모노드) 대입
        i /= 2;//부모 노드를 따라 올라감
    }
    h->heap[i] = item; //최종 i번 인덱스 위치에 item 삽입
}
```

6. 우선 heap_size 멤버 변수를 1 증가시키고, 이 값을 I 변수에 대입한다.

7. 인덱스가 1 번이 아니고, compare 함수를 호출하여 얻은 결과 값으로, 새로 삽입하려는 문자열이 i/2 번(부모 인덱스) 노드의 문자열보다 사전 순으로 앞에 위치한다면, 해당 i/2 번 인덱스의 값을 i 번 인덱스의 값에 대입한다. 부모 노드가 자식 노드의 위치로 이동하게 되는 것이다. i 의 값은 2 로 나누어져 다시 부모 노드를 따라 올라가며, 위의 과정을 반복하도록 한다.

8. 마지막으로, 최종 i 번 인덱스의 위치에 item 을 삽입하면 문자열을 키 값으로 하는 최소 히프에 하나의 요소 삽입이 완성된다.

다음은 문자열을 키 값으로 하는 최소 히프의 루트 노드를 삭제하여 반환하는 동작을 하는 delete_min_heap 함수이다. 루트 노드에는 히프의 마지막 요소를 대입한 뒤, 자식 노드 중 더 작은 값과 비교하여 더 큰 경우, 자식 노드와의 값을 비교하기를 반복하여 최소 히프 구조를 유지한다.

```
//·최소·히프의·루트·노드부터·삭제
element·delete_min_heap(HeapType*·h)·{
→   int·parent,·child;→//·부모/자식·노드·인덱스
→   element·item,·temp;→//·반환할·item,·임시·temp

→   item·=·h->heap[1];→//·반환할·요소는·1번·인덱스의·요소
→   temp·=·h->heap[(h->heap_size)--];→  //·마지막·요소를·temp에·저장하고,·힙·크기·1·감소
→   parent·=·1,·child·=·2;→//·부모,·자식·노드·인덱스는·1,2부터·시작
```

9. 부모, 자식 노드 인덱스 값을 갖는 parent 와 child 정수 변수를 선언한다. 반환할 루트 노드의 값을 저장할 item 변수, 그리고 마지막 노드의 값을 저장할 temp 를 선언한다.

10. 반환할 요소는 루트 노드이므로, 1 번 인덱스의 값을 item 에 대입한다. heap 의 heap_size 번 인덱스의 요소, 마지막 노드를 temp 구조체에 대입하고, heap_size 는 1 감소시키도록 한다. parent 와 child 의 값은 각각 1, 2 로 초기화한다.

```
→   //·child가·힙의·마지막·요소에·도달할·때까지·반복
→   while·(child·<=·h->heap_size)·{
→   →   //·힙의·마지막·요소가·아니고,·child(왼쪽·자식노드)가·child+1(오른쪽·자식·노드)보다·문자열이·뒤에·있는·것이라면
→   →   if·((child·<·h->heap_size)·&&·(compare(h->heap[child].name,·h->heap[child·+·1].name)·>·0))
→   →   →   child++;→   //·1증가하여·오른쪽·자식·노드와·비교하도록

→   →   //·temp의·문자열이·child·인덱스의·문자열보다·사전·순으로·앞에·있거나·같다면·반복문·종료
→   →   if·(compare(temp.name,·h->heap[child].name)·<=·0)
→   →   →   break;
```

11. child 의 값이 heap_size 보다 작거나 같을 때까지 while 반복문을 실행한다. child 가 heap_size 보다 작고, child(왼쪽 자식 노드)번 인덱스를 child+1(오른쪽 자식 노드)번 인덱스의 값을 비교하여, 왼쪽 자식 노드의 문자열이 오른쪽 자식 노드의 문자열보다 사전 순으로 뒤에 위치한 것이라면 child 의 값을 1 증가시킨다.

12. temp 의 name 문자열(마지막 노드의 값)이 child 번 인덱스의 name 문자열보다 사전 순으로 앞에 위치하거나 동일한 문자열이라면 반복문을 종료한다.

```
→   →   h->heap[parent]·=·h->heap[child];
→   →   parent·=·child;→//·부모의·인덱스는·자식의·인덱스로·이동
→   →   child·*=·2;→→   //·왼쪽·자식·노드·인덱스로·이동
→   }
→   h->heap[parent]·=·temp;→//·최종·부모·인덱스에·마지막·요소·값을·대입
→   return·item;
}
```

13. 그렇지 않은 경우라면, parent 번 인덱스에 child 번 인덱스의 값을 대입한다. 자식 노드가 부모 노드로, 위로 이동하게 되는 것이다. 이후 parent 에는 다시 child 의 값을 넣고, child 에는 2 를 곱하여, 한 단계 아래로 내려가 다시 이 과정을 반복하도록 한다.

14. while 반복문이 종료된 후, 마지막의 parent 번 인덱스에 temp 값을 대입하여 마지막 노드였던 값이 최소 히프의 적절한 위치에 삽입된다. 노드들의 교환이 성공적으로 이루어져 최소 히프의 구성이 마무리된 것을 의미한다. 마지막으로 item 의 값을 반환하고 나면, 이로써 루트 노드의 값이 성공적으로 삭제된다.

```
//·순서대로·히프의·요소·출력
void·display_heap(HeapType*·h)·{
→   int·i;
→   printf("<·히프·출력·>·\n");
→   for·(i·=·1;·i·<=·h->heap_size;·i++)·{
→   →   printf("%d:·%s·=>·",·i,·h->heap[i].name);
→   }
→   printf("\n\n");
}
```

15. 히프의 요소들을 배열의 순서대로 출력하는 display_heap 함수이다. count 의 값에 heap_size 를 대입하고, for 문을 돌리며 1 번 인덱스부터 count 의 값까지 각 배열 요소들을 출력하도록 한다.

main 함수 동작의 동작을 살펴보도록 하자.

```
int·main()·{
→   FILE*·fp;
→   element·tmp;→   →   →   //·임시·구조체
→   char·command,·name[20];→//·파일의·명령어·문자,·임시·문자·배열
→   int·i·=·0,·size·=·0,·max_count·=·0,·heap_size·=·0;→//·히프의·배열·크기·구하기
→   HeapType*·heap;
```

16. 파일 포인터 fp, 임시 구조체 tmp, 그리고 파일의 명령어 문자를 저장할 command 와 임시 문자 배열 name 을 선언한다. 반복문 제어 변수 i 와 히프의 배열 크기를 구하기 위해 사용되는 변수들 size, max_count, heap_size 를 0 으로 초기화하여 선언한다. 그리고 동적 할당할 히프 구조체를 가리키는 heap 포인터를 선언한다.

```
→   heap·=·create();→   //·히프·구조체·동적·할당
→   init(heap);→→   →   //·히프·초기화

→   fp·=·fopen("data.txt",·"rt");
→   if·(fp·==·NULL)error("파일·열기·실패");
```

17. create 함수를 호출하여 heap 에 히프의 구조체 메모리 공간을 동적 할당하고, init 함수를 호출하여 히프의 heap_size 멤버를 0 으로 초기화한다.

18. data.txt 파일을 읽기 모드로 연다.

```
//·파일·읽어·히프·노드의·개수(배열·크기)·결정
while·(!feof(fp))·{
    fscanf(fp,·"%c",·&command);
    if·(command·==·'i')·{
        fscanf(fp,·"%s",·name);
        size++;
    }
    else·if·(command·==·'o')·size--;
    //·input,·output·결과에·의한·최대·수용·가능한·배열의·길이·max_count·구하기
    if·(size·>·max_count)max_count·=·size;
}
rewind(fp);
```

19. 파일에서 데이터를 읽어 저장하기 전, 우선적으로 히프 배열에 삽입되는 요소의 개수 중 가장 큰 경우의 값을 구한다. data.txt 파일에 입장(input)을 의미하는 i 와 손님의 이름 또는 퇴장(output)알 의미하는 o 가 각 줄에 저장되어 있다. i 를 읽는 경우에는 size 를 1 증가시키고, o 를 읽는 경우에는 size 를 1 감소시켜 누적의 size 값 중, 가장 큰 경우의 size 를 max_count 에 저장한다. max_count 는 i 와 o 의 연산 결과에 의해, 배열의 길이가 가장 길었을 때의 경우의 값을 지니게 된다.

```
for·(i·=·1;i·<=·max_count;i·*=·2)
    heap_size·=·i;
heap_size·*=·2;→//·마지막·레벨에·모든·노드가·채워지도록·히프의·크기·결정
printf("heap_size·=%d·\n",·heap_size);

heap->heap·=·(element*)malloc(sizeof(element)·*·(heap_size));·//·히프의·배열·공간·동적·할당
```

29. max_count 의 값을 바탕으로, 히프 이진 트리를 구성할 때 마지막 레벨의 노드들까지 모두 포함한 배열의 크기를 정하기 위해 히프의 최종적 크기를 결정할 heap_size 값을 구한다. 마지막 레벨을 제외하고 완전 이진 트리의 레벨 1 부터 시작하여, 각 레벨의 노드들은 1 부터 2 배로 각각 증가한다. 높이 h 에 따라 전체 노드의 개수는 (2^h-1)이고, 레벨 I 의 노드의 개수는 (2*I -1)이다. 마지막 레벨의 노드들까지 고려하여 제어 변수 i 는 1 부터 max_count 에 도달할 때까지 2 씩 곱하여 지며 heap_size 에 i 를 대입한다. 마지막 레벨에 모든 노드들이 채워져 있음을 가정하고 배열의 길이를 결정해야 하므로, heap_size 는 for 반복문 이후에 마지막으로 2 가 한 번 더 곱해진다. 따라서, max_count 보다 큰 2 의 제곱 값을 지니게 된다.

30. heap 포인터에 결과로 얻게 된 heap_size 에만큼의 element 배열을 동적 할당한다.

```
//·다시·파일을·읽어·읽어·히프·삽입·또는·삭제
while·(!feof(fp))·{
    fscanf(fp,·"%c",·&command);
    if·(command·==·'i')·{
        fscanf(fp,·"%s",·name);
        printf(">>손님(%s)·입장·\n",·name);

        //·name·멤버에·문자·배열·동적·할당
        tmp.name·=·(char*)malloc(sizeof(char)·*·(strlen(name)·+·1));
        strcpy(tmp.name,·name);
        insert_min_heap(heap,·tmp);→//·최소·히프에·삽입
        display_heap(heap);→→    →   //·히프의·요소를·순서대로·출력
    }
```

31. 파일을 다시 읽어 명령어 문자에 따라 최소 히프에 삽입 또는 삭제를 한다. 명령어 문자가 'i'인 경우, 이름 문자열도 읽는다. tmp 구조체의 name 포인터에 읽은 name 문자열만큼의 문자 배열을 동적 할당하여 문자열을 복사한 뒤, insert_min_heap 를 호출하여 최소 히프에 삽입한다. 히프에 삽입한 후에 display_heap 을 호출하여 히프에 삽입되어 있는 요소들을 차례대로 출력한다.

```
    else·if·(command·==·'o')·{
        tmp·=·delete_min_heap(heap);→   //·최소·히프에서·삭제한·값을·tmp에·대입
        printf(">>손님(%s)·퇴장·\n",·tmp.name);
        free(tmp.name);→→   //·tmp의·name·문자·배열·메모리·해제
        display_heap(heap);
    }
}
```

32. 명령어 문자가 'o'인 경우, delete_min_heap 를 호출하여 삭제한 값을 tmp 에 대입한다. 최소 히프 삭제 연산이므로, 문자열이 사전 순으로 가장 앞에 위치한 것이 반환된다. tmp 에 저장된 손님의 이름 name 을 출력한 후에, 해당 문자 배열의 메모리를 해제한다. display_heap 함수를 호출하여 히프에 남아있는 요소들을 차례대로 출력한다.

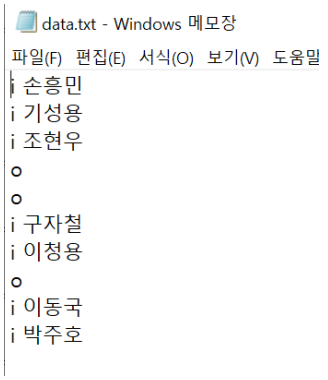
```
//·남아있는·배열·요소들의·문자·배열·메모리·해제
for·(i·=·1;i·<=·size;i++)
    free(heap->heap[i].name);
free(heap->heap);→  //·히프·배열·공간·메모리·해제
free(heap);→→    →   //·히프·구조체·메모리·해제

fclose(fp);→//·파일·닫기
return·0;
}
```

33. 마지막으로, 삭제한 후 남아있는 배열 요소들의 문자 배열을 메모리 해제한다. 여기서 size 는 위에서 파일을 처음 읽을 때 i 와 o 의 따라 연산한 마지막 값, 배열에 남아있는 요소의 개수이다.

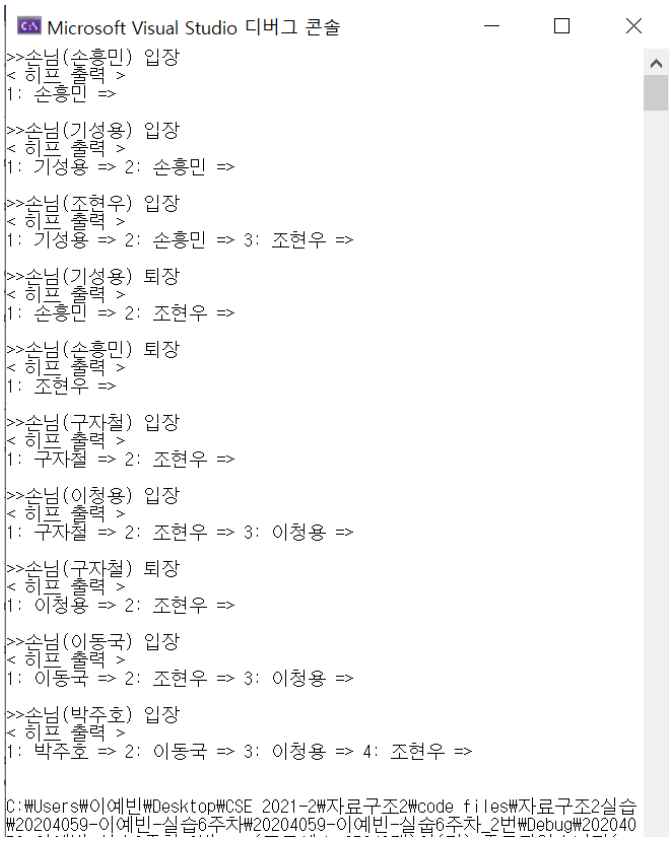
남아있는 요소들의 문자 배열을 메모리 해제하고 난 후, 히프 배열 공간의 메모리를 해제한다. 마지막으로 heap 구조체 메모리를 해제하고, 파일을 닫고 0 을 리턴하여 프로그램을 종료한다.

2.4 실행창



data.txt 파일은 위와 같이 입장(삽입) 또는 퇴장(삭제)를 뜻하는 명령어 문자 i 와 o, 그리고 손님의 이름이 저장되어 있다. 배열에 최대로 요소가 삽입되었을 때의 길이는 마지막 "박주호"가 삽입될 때의 길이, 총 4 이다. 4 개의 노드로 구성된 히프 이진 트리의 높이는 3 이고, 마지막 레벨 3 에 모든 노드들이 채워져 있음을 가정하면 총 7 개의 노드를 필요로 한다. 따라서 마지막 heap_size 는 4 의 값에서 2 가 곱해져 8 의 값을 가지게 되고, 배열의 길이는 총 8 이 된다.

결과창은 다음과 같다.



2.5 느낀점

2 번 문제를 풀면서 히프의 삽입과 삭제의 연산에 대한 이해도를 높일 수 있었다. 1 번 문제와 다르게 문자열을 키 값으로 하는 최소 히프를 구현해야 한다는 점에서, 문자열 처리 함수를 사용하여 문자열을 비교하여 얻는 반환 값을 바탕으로 노드들의 이동을 고려해야 했고, 때문에 삽입과 삭제의 연산 함수를 변경하는 과정이 이전보다는 복잡했던 것 같다. 또한 문자열을 동적 할당했기 때문에 이를 메모리 해제할 때에도 히프에 남아있는 노드들의 개수를 고려해야 했던 부분이 생각보다 간단하지 않았던 것 같다. 이러한 반복되는 고민의 과정 속에서 히프에 대한 이해도를 높일 수 있었다. 배열의 구조를 사용하지만, 이진 트리로 시각화하여 노드들의 이동과 교환을 한눈에 살펴볼 수 있다는 점에서, 히프는 매우 매력적인 자료구조인 것 같다.

3. 느낀점

이번 1번, 2번 문제를 통해 히프에 대한 이해도가 높아졌다. 우선순위에 따라서 최대값을, 또는 최소값을 반환하는 최대 히프, 최소 히프를 구현하여 히프에 대한 삽입과 삭제 연산 과정을 매우 흥미롭게 공부할 수 있었던 과제들이었던 것 같다.

처음에는 그 개념이 매우 생소하기도 했고, 복잡하게 보이는 알고리즘을 하나하나 이해하는 것이 쉽지 않았는데, 이번 과제를 해결하는 과정에서 배열 그리고 이진 트리로 삽입과 삭제의 각 과정을 표현하여 이해하려 노력해보니 이제는 히프의 개념에 대해 많이 익숙해진 것 같다. 시간이 오래 걸리더라도, 프로그램을 하나씩 디버깅 해가며 변수들의 변화를 확인하고, 또 직접 그림을 그려보며 삽입과 삭제의 과정을 직접 수행해 보는 것이 시간이 오래 걸리더라도 자료 구조에 대한 이해도를 높일 수 있는 가장 중요한 방법 중 하나라는 것을 새로 깨달았다.