

# 13

## Heap ADT

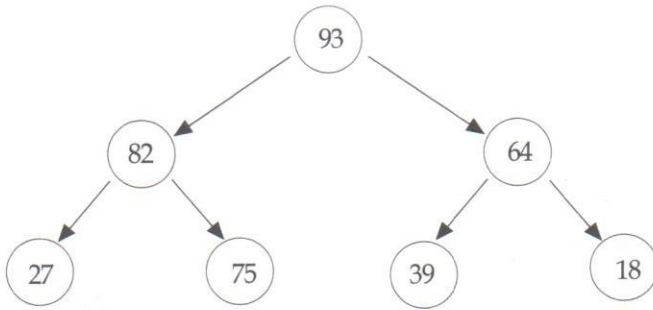
### 목적

이번 실습에서 여러분은

- 트리의 배열 표현을 사용하여 Heap ADT 구현을 작성한다.
- 여러분의 heap 클래스에서 우선순위 큐 클래스를 구현하기 위해 상속을 사용한다. 우선순위 큐를 사용하여 오퍼레이팅 시스템의 작업 스케줄러에 관한 모의실험을 개발한다.
- Heap ADT에 관한 여러분의 구현을 사용한 heap 생성 기술에 근거한 heap sort 함수를 만든다.
- Heap내에 위치한 다양한 우선순위를 갖는 원소가 있는 곳을 분석한다.

### 개요

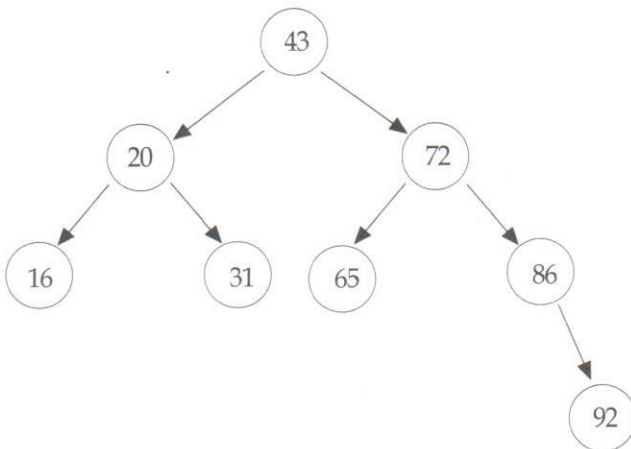
링크드 구조만이 트리들을 표현할 수 있는 유일한 방법은 아니다. 여러분이 아래에 주어진 이진 트리를 가지고 내용을 레벨 순서로 배열에 저장한다면 여러분은 다음의 배열을 생성한다.



번호	값
0	93
1	82
2	64
3	27
4	75
5	39
6	18

트리내에서 위치들 간의 관계와 배열내의 원소(entires)를 검사할 때 여러분은 원소가 배열의 순서(entry)  $N$ 에 적재되면 원소의 왼쪽 자식(child)은 순서(entry)  $(N-1) \bmod 2$ 가 되고 오른쪽 child는 순서(entry)  $2N+2$ 로 적재되고 부모는 순서 $(N-1) \bmod 2$ 로 적재된다. 이들의 사상은 부모에서 자식으로 이동하는 트리에서 쉽게 이동할 수 있다.

여러분은 이 사상을 이진 탐색 트리 ADT의 배열을 기본으로 하는 구현을 지원하는데 사용한다. 그러나 배열의 많은 부분이 사용되지 않은 트리 표현이 된다(여러분의 배열에서 dash로 표시된다).

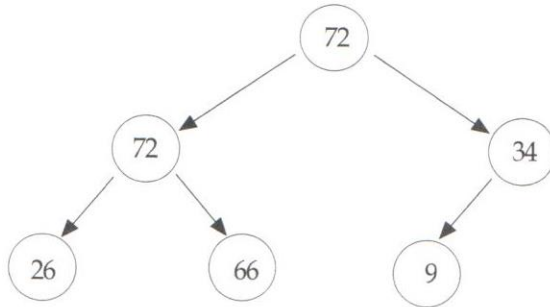


번호	값
0	43
1	20
2	72
3	16
4	31
5	65
6	86
7	-
8	-
9	-
10	-
11	-
12	-
13	-
14	92

이 실습에서 여러분은 heap이라고 불리는 트리의 다른 형태에 중점을 둔다. Heap은 다음 조건을 만족하는 이진 트리이다.

- 트리는 완전트리이다. 즉 트리 내의 모든 레벨은 가득 차 있다. 바닥 레벨을 제외하고 만일 바닥 레벨이 가득 차 있지 않으면 모든 오른쪽에서 손실 원소(missing element)들이 발생한다.
- 트리 내의 각 원소는 대응하는 우선순위 값을 갖는다. 각 원소 E에서 E의 모든 자손은 E의 우선순위와 같거나 작은 우선 순위들을 갖는다. 우선순위는 유일하지 않은 것에 주의하라.

이번 실습의 첫장에서 본 트리는 heap이다. 아래에서 보는 트리과 같이



트리가 완전하다는 의미는 heap이 레벨의 순서에 따라 중간에 빈 공간을(사용되지 않은) 알려주는 것 없이 배열에 저장되어질 수 있다는 의미이다. 결과는 여러분이 쉽게 가지의 위 아래로 쉽게 이동할 수 있는 간결한 heap이다.

명확하게 heap내의 다양한 원소의 우선순위 사이의 관계는 효과적인 검색 수행을 지원할 정도는 아니다. 관계는 간단하기 때문에 여러분은 가장 높은 우선순위(root) 원소를 다시 이동시킨 후 또는 새로운 원소를 삽입 후에 빠르게 재구성할 수 있다. 결과적으로 여러분은 빠르게 heap내의 원소를 우선순위에 기본을 둔 내림차순으로 수행할 수 있다. 간결한 배열 표현으로 조합되어진 이 우선순위는 heap을 우선순위 큐에 대해 이상적인 표현으로 만든다(실습 중 연습 1). 그리고 heap sort(실습 중 연습 2)라고 불리는 효과적인 정렬 알고리즘의 기본형태를 만든다.

## Heap ADT

### 원소

heap내의 원소들은 일반적인 HE 타입이다. 각 원소는 heap내에서 원소의 관계위치를 결정하는데 사용되는 우선순위를 갖는다. 원소들은 보통 추가 데이터를 포함한다. 우선순위가 유일하지 않다는 것에 주의하라. 몇몇 원소들은 아주 같게 같은 우

선순위를 갖는다. 타입 HE는 원소의 우선순위를 반환하는 `pty()`라는 함수를 제공한다. 여러분은 반드시 여섯 개의 기본 관계 기능을 이용하여 우선순위를 비교해야 한다.

## 구조

원소들은 완전 이진 트리형태를 갖는다. 트리내에서 각 E원소에 대해 모든 E의 자손은 E의 우선순위와 같거나 작은 우선순위를 갖는다.

## 기능

**Heap (int maxNumber = defMaxHeapSize)**

요구사항:

없음.

결과:

객체 생성자. 빈 heap을 만든다. MaxNumer 원소들을 갖는 heap에 대해 충분한 메모리를 할당한다.

**~Heap ( )**

요구 사항:

없음.

결과:

객체 파괴자. heap을 적재하는데 사용된 메모를 할당해제한다.

**void insert (const HE &newElement)**

요구사항:

heap은 가득 차 있지 않다.

결과:

원소를 heap에 삽입한다. heap내의 바닥 가장 오른쪽에 이 원소를 삽입하고 재 적재된 heap을 정의하는 우선순위까지 위로 이동시킨다.

**HE removeMax ( )**

요구사항:

heap은 비어 있지 않다.

**결과:**

가장 높은 우선순위(root)를 갖는 원소를 heap에서 제거하고 그 원소를 반환한다. 바닥 가장 오른쪽 원소로 루트 원소를 대체하고 재 저장된 heap을 정의하는 우선 순위까지 아래로 그 원소를 이동시킨다.

**void clear ( )****요구사항:**

없음.

**결과:**

heap의 모든 원소를 제거한다.

**int empty ( ) const****요구사항:**

없음.

**결과:**

heap이 비어 있으면 1을 반환하고 그렇지 않으면 0을 반환한다.

**int full ( ) const****요구사항:**

없음.

**결과:**

heap이 가득 차 있으면 1을 반환하고 그렇지 않으면 0을 반환한다.

**void showStructure ( ) const****요구사항:**

없음.

**결과:**

heap과 배열과 트리형태에 있는 원소들의 우선순위를 출력한다. 트리는 왼쪽(root)에서 오른쪽(leaves)으로 향하는 가지를 가지고 출력된다. 즉 트리는 원래의 방향에서 시계 반대방향으로 90도 회전해서 출력된다. 만일 heap이 비어 있으면 "Empty heap"를 출력한다. 이 기능은 단지 시험과 디버깅을 목적으로 한다.

---



## 실습 13 : 표지

날짜 \_\_\_\_\_ 구분 \_\_\_\_\_  
 성명 \_\_\_\_\_

강사가 여러분에게 할당한 다음에 연습문제들에 할당된 열에 체크표시를 하고 이 겹장을 앞으로의 연구에서 여러분이 제출할 과제물 앞에 붙이시오.

	할 일	완 료
실습 전 연습	✓	
연 결 연습	✓	
실습 중 연습 1		
실습 중 연습 2		
실습 중 연습 3		
실습 후 연습 1		
실습 후 연습 2		
	총 점	

## 실습 13 : 실습 전 연습

날짜 \_\_\_\_\_ 구분 \_\_\_\_\_  
 성명 \_\_\_\_\_

1단계 : Heap ADT에서 heap의 배열 표현을 사용하는 기능을 구현하여라. heap은 다른 크기를 갖을 수 있다. 따라서 여러분은 heap 원소들(element)에 따라 heap이 가질 수 있는 원소의 최대 수(maxSize)를 적재할 필요가 있고 실제 사용하는 원소의 수(size)를 적재할 필요가 있다. 파일 *heap.h*에 있는 다음의 선언에 기본으로 하여 구현하라. Show structure기능의 구현은 파일 *show13.cpp*에 있다.

```
const int defMaxHeapSize = 10; // Default maximum heap size

template <class HE>
class Heap
{
public:

    // Constructor
    Heap (int maxNumber = defMaxHeapSize);

    // Destructor
    ~Heap ( );

    // Heap manipulation operations
    void insert (const HE &newElement); // Insert element
    HE removeMax ( ); // Remove max pty element
    void clear ( ); // Clear heap

    // Heap status operations
    int empty ( ) const; // Heap is empty
    int full ( ) const; // Heap is full

    // Output the heap structure -- used in
    // testing/debugging
```

```

void showStructure ( ) const;

private:
    // Recursive partner of the showStructure( ) function
    void showSubtree (int index, int level) const;

    // Data members
    int maxSize,    // Maximum number of elements in the heap
        size;      // Actual number of elements in the heap
    HE *element;   // Array containing the heap elements
};

```

**2단계 :** 파일 *heap.cpp*에 여러분의 Heap ADT의 구현을 저장하라. 여러분의 코드를 문서화하여라.




## 실습 13 : 연결 연습

날짜 \_\_\_\_\_ 구분 \_\_\_\_\_

성명 \_\_\_\_\_

여러분의 강사와 함께 여러분이 이 연습을 실습 기간 이전이나 실습 기간 중에 마칠 수 있는지 체크하라.

 파일 `test13.cpp`에 있는 시험 프로그램에서 여러분은 다음 명령어를 사용하여 Heap ADT의 구현을 상호적으로 시험할 수 있다.

명령어	기 능
+pty	지정된 우선순위를 갖는 원소를 삽입.
-	heap에서 가장 높은 우선순위를 갖는 원소를 삭제하고 출력한다.
E	heap이 비어 있는지 조사한다.
F	heap이 가득 차 있는지 확인한다.
C	heap을 clear한다.
Q	시험 프로그램을 종료한다.

**1단계 :** Heap ADT의 구현에 대한 시험 계획을 준비하라. 여러분의 시험 계획은 비거나 가득 차거나 단일 원소를 갖는 heap를 포함해서 다양한 크기들의 heap를 모두 다루어야 한다.

**2단계 :** 여러분의 시험 계획을 실행하라. 여러분의 구현에서 실수를 발견하면 수정하고 다시 실행하라.

Heap ADT 기능 시험 계획			
시험 항목	명령어	예상 결과	검사