

데이터구조 Assignment #4

2272027 전자전기공학 심희윤

코드 설명(부분마다 각각 설명을 추가)

```
#include <iostream>
#include <vector>
#include <stdexcept>
```

```
using namespace std;
```

```
// 2272027 전자전기공학과 심희윤
```

```
class MaxHeap { // MaxHeap 방식으로 만들어지는 vector에 대한 클래스
```

- MaxHeap 방식으로 만들어지는 vector에 대한 클래스로, 문제에서 현재 입력된 데이터 중 가장 큰 값을 출력하는 데에 사용된다.

```
private:
```

```
    std::vector<int> heap; // 실제로 데이터가 저장되는 vector
```

```
    void heapifyUp(int index) { // MaxHeap.push()가 수행되었을 때 호출되는 함수
        // 밑 쪽 index에서부터 시작해 루트(index 0) 또는 적절한 자리의 index를 찾을
        때까지
```

```
        // 부모 노드 값과 비교하며 heap 트리의 성질에 맞게 위로 올라가면서 값의
        자리를 바꾸어주는 함수
```

```
        while(index > 0){ // 기준 노드가 루트에 도달하기 전까지 반복
            int parent = (index-1)/2; // vector에서의 부모 노드 위치
            if(heap[parent] >= heap[index]) break;
            // 만약 부모 노드의 값이 기준 노드의 값보다 크거나 같을 경우 반복 종료
```

```
            swap(heap[parent], heap[index]);
            // 기준 노드의 값이 부모 노드보다 큰 경우 두 값을 바꾼다.
            index = parent;
            // 부모 노드가 새 기준 노드가 된다.
```

```
        }
```

```
    }
```

- heapifyUp() 함수는 트리에 새로운 값이 push되었을 때 MaxHeap의 무너진 성질을 복구하는 함수이다.
- 새로운 값은 항상 vector의 맨 뒤에 추가되기 때문에, 맨 뒤 노드를 기준으로 부모 노드와 비교하기 시작한다. 반복문으로 부모 노드와 기준 노드의 자리를 바꾸어가며 트리의 위쪽으로 이동하다가, 기준 노드의 값이 더 작은 경우나 / 기준 노드가 부모 노드가 없는 경우 반복을 멈춘다.
- 결과적으로 heap 트리의 루트 노드는 가장 큰 값을 가지게 된다.

```
void heapifyDown(int index) { // MaxHeap.popMax()가 수행되었을 때 호출되는 함수
    // 위 쪽 index에서부터 시작해 적절한 자리의 index를 찾을 때까지
    // 두 개의 자식 노드 값과 비교하며 heap 트리의 성질에 맞게 아래로 내려가며
    // 값의 자리를 바꾸는 함수
```

```
    int largest = index;
    // largest 는 기준 노드 / 왼쪽 자식 노드 / 오른쪽 자식 노드 중
    // 가장 큰 값을 가진 노드의 vector index를 저장하는 변수
```

```
    int left = index*2+1;
    int right = index*2+2;
    // 자식 노드 index 계산
```

```
    // 왼쪽 자식이 존재하고, largest 노드 값보다 왼쪽 자식의 값이 더 큰 경우
    if (left < heap.size() && heap[left] > heap[largest]) {
        largest = left;
    }
```

```
    // 오른쪽 자식이 존재하고, largest 노드 값보다 오른쪽 자식의 값이 더 큰 경우
    if (right < heap.size() && heap[right] > heap[largest]) {
        largest = right;
    }
```

```
    // 루트가 최대값이 아니라면 자식 노드와 자리를 바꾼다.
    if (largest != index) {
        swap(heap[index], heap[largest]);
        heapifyDown(largest); // 자리를 바꾼 자식 노드 기준으로 heapifyDown()을
        재귀호출한다.
    }
}
```

- heapifyDown() 함수는 트리에서 루트 노드가 pop되고, 루트 노드 자리에 맨 끝 노드를 옮겼을 때 MaxHeap의 무너진 성질을 복구하는 함수이다.
- 루트 노드를 기준으로 자식 노드들과 값을 비교하기 시작한다. 반복문으로 더 값이 큰 자식 노드와 기준 노드의 자리를 바꾸어가며 트리의 아래쪽으로 이동하다가, 기준 노드의 값이 자식 노드들의 값보다 더 큰 경우나 / 기준 노드가 자식 노드가 없는 경우 반복을 멈춘다.
- 결과적으로 heap 트리의 루트 노드는 가장 큰 값을 가지게 된다.

public:

```
// heap에 새로운 value 넣기
void push(int value) {
    heap.push_back(value);
    // heap vector의 맨 뒤에 값을 넣는다.
    heapifyUp(heap.size()-1);
    // 맨 뒤의 노드 기준으로 heapifyUp()을 호출하여 망가진 heap 트리 성질을
    복구한다.
}
```

- heap 트리에 value 값을 가지는 새 노드를 삽입하는 함수이다. 새로 삽입한 노드로 인해 heap 트리 성질이 무너졌을 것을 감안해 삽입한 노드 기준으로 heapifyUp()을 호출한다.

```
// maximum값 얻기
int getMax() const {
    if(isEmpty()) return -1;
    // 데이터가 없을 때의 예외처리

    return heap[0];
    // MaxHeap 구조에서는 vector의 맨 앞 값이 가장 큰 값이다.
}
```

- MaxHeap 구조에서는 루트 노드가 가장 큰 값을 가진다.(배열이므로 index 0)

```
// heap에서 maximum값 지우기
int popMax() {
    if(isEmpty()) return -1;
    // 데이터가 없을 때의 예외처리

    int max = heap[0];
    heap[0] = heap.back();
    heap.pop_back();
}
```

```
// 맨 앞 값이 가장 큰 값이므로 max 변수에 넣고, 맨 앞에 맨 뒤 값을 옮긴다.
```

```
if(!heap.empty()) heapifyDown(0);
```

```
// 데이터가 있는 경우 맨 앞의 노드 기준으로 heapifyDown()을 호출하여 망가진 heap 트리 성질을 복구한다.
```

```
return max;
```

```
// 가장 큰 값을 반환한다.
```

```
}
```

- heap 트리에서 가장 큰 값을 가지는 노드 즉 루트 노드를 pop 하는 함수이다. 이후 빈 루트 노드 자리에 맨 끝에 있는 노드를 옮긴다. 이로 인해 heap 트리 성질이 무너졌을 것을 감안해 루트 노드 기준으로 heapifyDown()을 호출한다.

```
// Check if the heap is empty
```

```
bool isEmpty() const {
```

```
    return heap.empty();
```

```
    // heap이 비어있는지 아닌지 확인한다.
```

```
}
```

- heap이 비어있는 경우의 예외처리를 위한 함수

```
};
```

```
int main() {
```

```
    MaxHeap maxHeap;
```

```
    int sum = 0;
```

```
    std::vector<int> numbers = {17514, 3638, 31955, 3821, 5867, 12996, 6653, 3173};
```

```
    for (int num : numbers) {
```

```
        maxHeap.push(num);
```

```
        sum += maxHeap.getMax();
```

```
        std::cout << "Added order: " << num << " | Current Max: " <<
```

```
maxHeap.getMax() << " won\n";
```

```
    }
```

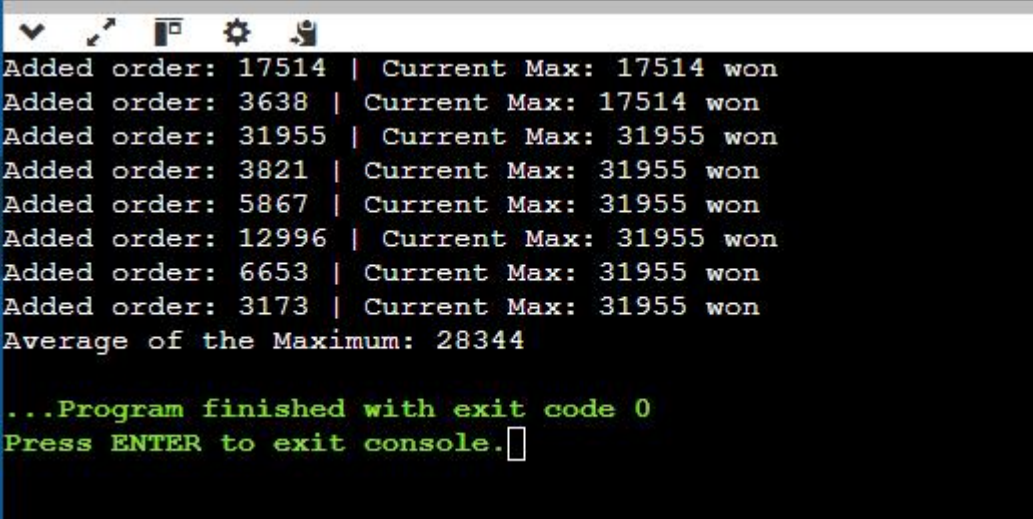
```
    std::cout << "Average of the Maximum: " << sum / numbers.size();
```

```
    // 각 회마다 구한 가장 큰 값들의 평균값을 출력한다.
```

```
    return 0;
}
```

- numbers 벡터에 있는 데이터들을 하나씩 maxHeap에 push하면서 현재 삽입한 데이터들 중 가장 큰 값을 sum에 더하고, 최종적으로 각 반복 회차마다 구한 가장 큰 값들의 평균값을 출력하는 함수이다.

결과 설명



```
Added order: 17514 | Current Max: 17514 won
Added order: 3638 | Current Max: 17514 won
Added order: 31955 | Current Max: 31955 won
Added order: 3821 | Current Max: 31955 won
Added order: 5867 | Current Max: 31955 won
Added order: 12996 | Current Max: 31955 won
Added order: 6653 | Current Max: 31955 won
Added order: 3173 | Current Max: 31955 won
Average of the Maximum: 28344

...Program finished with exit code 0
Press ENTER to exit console.
```

number 벡터의 값을 하나씩 maxHeap에 push하면서 그 때마다의 maxHeap에서 가장 큰 값을 출력한다.

Maximum 값들의 평균 : $((17514 * 2) + (31955 * 6)) / 8 = 28344.75 \Rightarrow 28344$