```
*********************************************************************************************
* Name : 차윤성
* Student ID : 20163162
* Program ID : Hw#5
* Description : 키보드로부터 heap의 max size와 입력받을 데이터의 개수, 데이터를 입력받은 후, 입력
*               받은 데이터를 base로 Min Heap을 구현하는 프로그램이다.
* Algorithm
* - 먼저 heap의 max size와 데이터의 개수, 데이터를 입력받는다.
* - 입력받은 max size를 토대로 heap과 binary tree를 구현한다. binary tree는 배열로 구현한다.
* - binary tree는 max_size와 배열을 참조할 index 변수를 갖는다.
* - binary tree의 max_size는 입력받은 데이터의 개수가 된다.
* - 먼저 입력받은 데이터 하나하나를 insertTree를 통해 Tree에 삽입한다. index 변수를 사용해 값을
*   넣는다. 값을 넣은 후 index값을 하나 올려준다.
* - tree에 값이 다 insert 되면 printTree를 통해 binary tree를 출력한다.
* - getTree는 tree의 값을 하나씩 반환하는 함수로, 먼저 zeroIndex를 통해 index를 0으로 초기화 시켜
*   준 후 binary tree의 가장 앞 index부터 하나하나 값을 뽑는다.
* - binary tree로부터 산출한 값은 heap에 insert 해준다.
* - heap또한 max_size와 index 변수를 가지며, heap이라는 이름의 배열을 갖는다.
* - heap의 max_size는 입력받은 max_size의 값으로 결정되며, 초기 index의 값은 0이다.
* - 먼저 isFull과 isEmpty는 각각 index가 max_size일때, 0일때 true를 반환해주는 함수이다.
* - Heap에 값을 insert시에 먼저 heap이 isFull인지 아닌지 검사한다.
* - 아닐 경우 먼저 지역변수 i를 ++index로 초기화 시킨다. 이후 삽입하려는 값과 부모를 검사한다.
* - 검사하여 부모, 즉 heap[i/2]보다 삽입하려는 값이 작고 i가 1이 아닐 때 까지 부모를 현재 i번째
*    index로 옮겨준다.
* - 반복 검사하여 조정이 완료되면 마지막으로 설정된 i번째 index에 삽입하려는 값을 넣어준다.
* - Min Heap에서의 delete는 가장 최상위 부모, 즉 가장 작은 값을 delete하게 된다.
* - 먼저 Heap이 isEmpty인지 아닌지 검사한다.
* - Heap에 한 개라도 값이 있다면, parent, child, temp, item이라는 int형 변수를 선언 후 item에는
*   최상의 부모의 값, temp에는 heap에서의 가장 마지막 값, parent는 1, child는 2로 초기화해준다.
*   특히 temp에 heap에 마지막 값을 넣어줄 때, delete를 완료하면 값이 하나 감소하므로 index를 하
*   나 줄여준다.
* - parent는 부모 index를 의미하고, child는 자식 index를 의미한다.
* - 먼저 child값이 index 보다 작으면 현재 부모 index의 자식들을 비교하여 더 작은 자식 값을 찾는다.
* - 더 작은 자식 index의 값을 부모 index 위치로 옮겨주고, parent를 child로 초기화 시키고, child*2
*   를 해주어 다음 자식들을 검사한다.
* - 마지막으로 heap의 마지막값이 저장된 temp보다 마지막으로 검사한 heap의 child 값이 더 크면 반
*   복문을 종료시킨 후, temp의 값을 마지막으로 검사한 child의 부모 index로 넣어준다.
* - heap의 level은 현재 heap에서 마지막 값이 저장된 index를 log2를 취해준 다음 1을 더해주면 된다.
* Variables
* - tree : max_size - tree의 최대 size를 저장하는 변수로, 입력받을 데이터 값을 max_size로 삼는다.
* - tree : index - tree에서 현재 참조 위치를 저장하는 변수이다.
* - tree : *arr - 입력받은 값을 저장할 배열 변수이다.
* - Tree(int size) - Tree 객체를 만드는 생성자 함수로, tree의 크기는 size만큼 할당된다.
* - insertTree(int value) - 매개변수 value를 Tree에 삽입하는 함수이다.
* - plusIndex() - tree객체의 index변수를 하나 더해주는 함수이다.
* - zeroIndex() - tree객체의 index변수를 0으로 초기화해주는 함수이다.
* - getTree() - tree객체에서 index변수가 참조하는 값을 반환해주는 함수이다.
* - heap : max_size - heap의 최대 size를 저장하는 변수로, size의 값을 max_size로 삼는다.
* - heap : index - heap에서 마지막으로 들어가 있는 데이터의 위치를 저장해주는 index 변수이다.
* - heap : *heap - data를 저장하는 배열 변수이다.
* - isFull() - heap의 데이터가 max_size만큼 가득 차 있는지 검사하는 함수이다.
* - isEmpty() - heap에 데이터가 하나도 없는지 검사하는 함수이다.
* - insertHeap() - heap에 데이터를 삽입하는 함수이다.
* - deleteHeap() - Min heap에서 가장 작은 data, 최상위 부모를 제거하고 반환하는 함수이다.
* - printHeap() - 현재 heap에 들어있는 모든 data를 출력하는 함수이다.
* - size - 키보드로부터 입력받은 heap의 max size의 값을 저장하는 int형 변수이다.
* - amountInput - 키보드로부터 최초로 입력받을 데이터의 개수를 저장하는 int형 변수이며, binary
*                 tree의 max size 또한 같은 값이다.
* - input - 키보드로부터 최초로 입력받고, tree에 insert될 data가 저장되는 int형 변수이다.
*********************************************************************************************
```

```
#include <iostream>

using namespace std;
*********************************************************************************************
* class : Tree
* description : Tree는 binary tree를 구현하고, 관련 ADT를 가지는 객체이다.
* variables
* - max_size : tree의 size가 저장되는 변수이다.
* - index : tree가 현재 참조하고 있는 index값이 저장되는 변수이다.
* - *arr : tree에 있는 값이 저장되는 배열변수이다.
*********************************************************************************************
class Tree {
private:
        int max_size;
        int index;
        int *arr;
public:
        Tree(int size) {
                max_size = size;
                arr = new int[size];
                index = 0;
        }
        void insertTree(int value);
        void plusIndex();
        void zeroIndex();
        void printTree();
        int getTree();
};
*********************************************************************************************
* function : insertTree
* description : insertTree의 사용 목적은 Tree에 입력받은 data값을 삽입하는 것이다.
*********************************************************************************************
void Tree::insertTree(int value) {
        arr[index] = value;
        plusIndex();
}
*********************************************************************************************
* function : plusIndex
* description : plusIndex의 사용 목적은 private으로 설정되어있는 index를 하나 올려주는 것이다.
*********************************************************************************************
void Tree::plusIndex() {
        index++;
}
*********************************************************************************************
* function : zeroIndex
* description : zeroIndex의 사용 목적은 private으로 설정되어 있는 index를 0으로 초기화 시키는 것
*               이다.
*********************************************************************************************
void Tree::zeroIndex() {
        index = 0;
}
*********************************************************************************************
* function : printTree
* description : printTree의 사용 목적은 현재 Tree에 삽입되어 있는 값을 출력하는 것이다.
*********************************************************************************************
void Tree::printTree() {
        cout << "( ";
        for (int i = 0; i < max_size; i++) {
                cout << arr[i] << " ";
        }
        cout << ")";
}
```

```
*********************************************************************************************
* function : getTree
* description : getTree의 사용 목적은 현재 index가 참조하고 있는 data를 반환하는 것이다.
* variables
* - result - 반환할 tree의 값을 저장하는 int형 변수이다.
*********************************************************************************************
int Tree::getTree() {
	int result;
	result = arr[index];
	plusIndex();
	return result;
}
*********************************************************************************************
* class : Heap
* description : Heap은 Min heap을 구현하고 관련 ADT를 가지는 객체이다.
* variables
* - max_size : Heap의 size를 저장하는 int형 변수이다.
* - index : Heap에서 가장 마지막 data를 가리키는 index를 저장하는 int형 변수이다.
* - *heap : 구현된 heap의 데이터를 저장하는 배열변수이다.
*********************************************************************************************
class Heap {
private:
	int max_size;
	int index;
	int *heap;
public:
	Heap(int size) {
		max_size = size;
		heap = new int[max_size];
		index = 0;
	}
	bool isFull();
	bool isEmpty();
	void insertHeap(int element);
	int deleteHeap();
	int levelHeap();
	void printHeap();
};
*********************************************************************************************
* function : isFull
* description : isFull의 사용 목적은 heap이 가득 찬 상태인지 검사하는 것이다.
*********************************************************************************************
bool Heap::isFull() {
	if (index == max_size)
		return true;
	else
		return false;
}
*********************************************************************************************
* function : isEmpty
* description : isEmpty의 사용 목적은 heap이 비어있는 상태인지 검사하는 것이다.
*********************************************************************************************
bool Heap::isEmpty() {
	if (index == 0)
		return true;
	else
		return false;
}
```

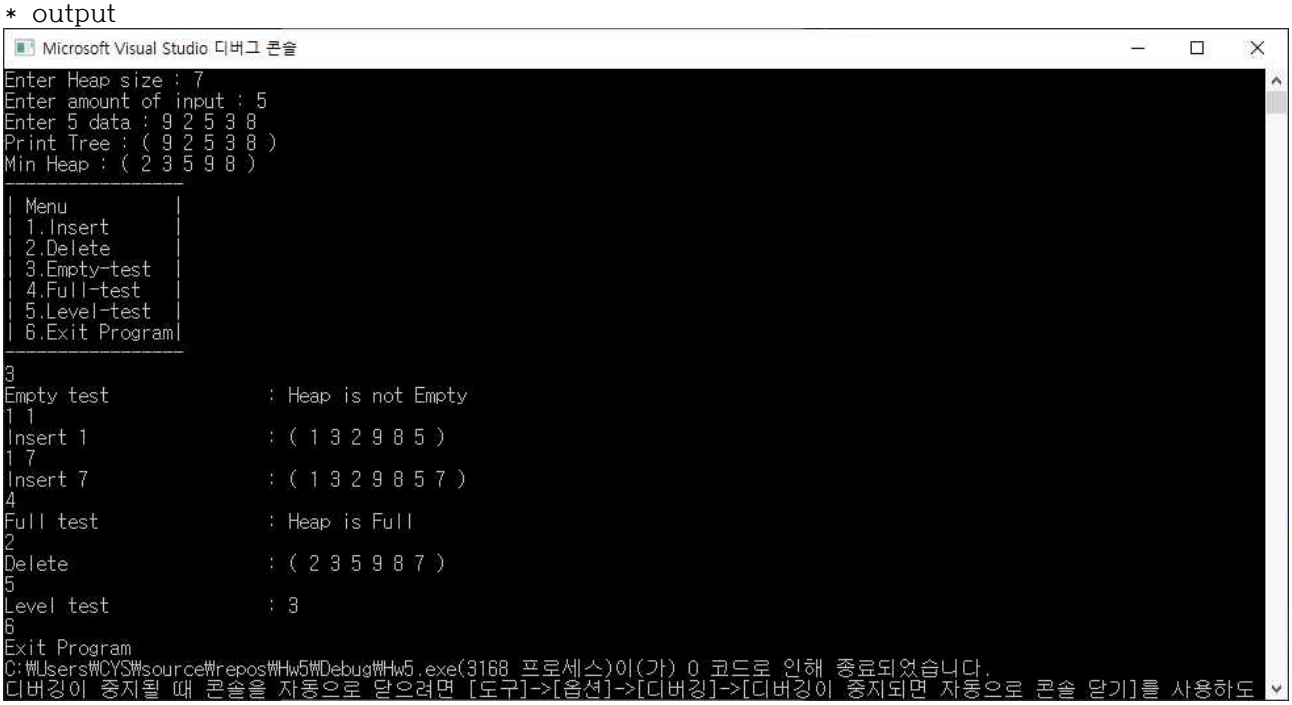
```
*****************************************************************************************
* function : insertHeap
* description : insertHeap의 사용목적은 매개변수 element를 heap에 삽입하는 것이다.
* variables
* - i : heap에 data를 삽입하고, 부모의 값과 비교하기 위한 index를 저장하는 변수이다.
*****************************************************************************************
void Heap::insertHeap(int element) {
        int i;
        if (isFull()) {
                cout << "Heap is Full" << endl;
                return;
        }
        i = ++index;
        while ((i != 1) && (element < heap[i / 2])) {
                heap[i] = heap[i / 2]; // 부모보다 값이 작으면
                i = i / 2;        // 부모를 현재 index로 옮김
        }
        heap[i] = element;                // 조정이 완료되면 element 삽입
}

*****************************************************************************************
* function : deleteHeap
* description : deleteHeap의 사용 목적은 Min Heap에서 가장 작은 값을 제거하고 반환하는 것이다.
* variables
* - parent : 검사하려는 부모의 index를 저장하는 int형 변수이다.
* - child : 검사하려는 자식의 index를 저장하는 int형 변수이다.
* - temp : heap에서 가장 마지막에 있는 data를 저장하기 위한 int형 변수이다.
* - item : heap에서 제거되는 가장 작은 값을 저장하는 int형 변수이다.
*****************************************************************************************
int Heap::deleteHeap() {
        if (isEmpty()) {
                cout << "Heap is empty" << endl;
                return 0;
        }

        int parent, child, temp, item;

        item = heap[1]; // 최상위 부모의 값을 저장
        temp = heap[index--];  //가장 마지막 값을 저장
        parent = 1;      // 부모 인덱스를 의미
        child = 2;               // 자식 인덱스를 의미

        while (child <= index) {
                if ((child < index) && (heap[child] > heap[child + 1]))
                        child++;          // 더 작은 자식값을 찾음
                else if (temp <= heap[child])
                        break;
                heap[parent] = heap[child];       // 작은 자식값을 부모 위치로 변경
                parent = child;
                child = child * 2;
        }
        heap[parent] = temp; // heap의 마지막 값을 부모자리에 삽입
        return item;
}
*****************************************************************************************
* function : levelHeap
* description : levelHeap의 사용 목적은 heap의 level이 몇인지 검사하는 것이다.
*****************************************************************************************
int Heap::levelHeap() {
        return int(log2(index)+1);
}
```

```
*****************************************************************************************
* function : printHeap
* description : printHeap의 사용 목적은 현재 Heap에 있는 모든 data를 출력하는 것이다.
*****************************************************************************************
void Heap::printHeap() {
	cout << "( ";
	for (int i = 1; i < index + 1; i++) {
		cout << heap[i] << " ";
	}
	cout << ")" << endl;
}

int main() {
	int size;
	int amountInput;
	int input;

	cout << "Enter Heap size : ";
	cin >> size;

	cout << "Enter amount of input : ";
	cin >> amountInput;

	Heap h1(size);
	Tree tree(amountInput);

	cout << "Enter " << amountInput << " data : ";

	for (int i = 0; i < amountInput; i++) {
		cin >> input;
		tree.insertTree(input);
	}

	cout << "Print Tree : ";

	tree.printTree();
	cout << endl;

	tree.zeroIndex();
	// insert하면서 증가된 index를 0으로 초기화시킴

	for (int i = 0; i < amountInput; i++) {
		h1.insertHeap(tree.getTree());
		// tree에서 값을 추출해 heap에 insert
	}

	cout << "Min Heap : ";

	h1.printHeap();

	cout << "-----------------" << endl;
	cout << "| Menu\t\t|" << endl;
	cout << "| 1.Insert\t|" << endl;
	cout << "| 2.Delete\t|" << endl;
	cout << "| 3.Empty-test\t|" << endl;
	cout << "| 4.Full-test\t|" << endl;
	cout << "| 5.Level-test\t|" << endl;
	cout << "| 6.Exit Program|" << endl;
	cout << "-----------------" << endl;
```

```
	while (1) {		// 6 또는 그 외의 값이 입력되면 프로그램이 종료됨.
		cin >> input;
		switch (input) {
		case 1:
			int data;
			cin >> data;
			h1.insertHeap(data);
			cout << "Insert " << data << "\t\t : ";
			h1.printHeap();
			break;
		case 2:
			h1.deleteHeap();
			cout << "Delete\t\t\t : ";
			h1.printHeap();
			break;
		case 3:
			cout << "Empty test\t\t : ";
			if (h1.isEmpty())
				cout << "Heap is Empty" << endl;
			else
				cout << "Heap is not Empty" << endl;
			break;
		case 4:
			cout << "Full test\t\t : ";
			if (h1.isFull())
				cout << "Heap is Full" << endl;
			else
				cout << "Heap is not Full" << endl;
			break;
		case 5:
			cout << "Level test\t\t : ";
			cout << h1.levelHeap() << endl;
			break;
		default :
			cout << "Exit Program";
			return 0;
			break;
		}
	}
}
```

****

* output



****