Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Ba, 8 tháng 4 2025, 1:33 PM
Kết thúc lúc	Thứ Ba, 15 tháng 4 2025, 12:32 AM
Thời gian thực hiện	6 Các ngày 10 giờ
Điểm	7,00/7,00
Điểm	**10,00** trên 10,00 (**100**%)

Câu hỏi 1

Đúng

Đạt điểm 1,00 trên 1,00

Implement functions: **Peek, Pop, Size, Empty, Contains** to a maxHeap. If the function cannot execute, **return -1**.

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"
template<class T>
class Heap {
protected:
    T* elements;
    int capacity;
    int count;
public:
    Heap()
    {
        this->capacity = 10;
        this->count = 0;
        this->elements = new T[capacity];
    }
    ~Heap()
    {
        delete[]elements;
    }
    void push(T item);

    bool isEmpty();
    bool contains(T item);
    T peek();
    bool pop();
    int size();

    void printHeap()
    {
        cout << "Max Heap [ ";
        for (int i = 0; i < count; i++)
            cout << elements[i] << " ";
        cout << "]\n";
    }
private:
    void ensureCapacity(int minCapacity);
    void reheapUp(int position);
    void reheapDown(int position);
};
//Your code goes here
```

For example:

Test	Result

`Heap<int> maxHeap;`
`for (int i=0;i<10;i++){`
`  maxHeap.push(i);`
`}`
`cout << maxHeap.size();`
	10

Test	Result

Heap<int> maxHeap;
for (int i=0;i<10;i++){
maxHeap.push(i);
}
cout << maxHeap.isEmpty();
	0

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
// Kiểm tra heap có rỗng không
template<class T>
bool Heap<T>::isEmpty() {
    return count == 0;  // Nếu count = 0 thì heap rỗng
}

// Trả về số lượng phần tử trong heap
template<class T>
int Heap<T>::size() {
    return count;  // count lưu số lượng phần tử hiện tại
}

// Trả về phần tử lớn nhất (không xóa nó)
template<class T>
T Heap<T>::peek() {
    if (count == 0) return (T)-1;  // Nếu heap rỗng, trả về -1
    return elements[0];  // Phần tử lớn nhất luôn ở vị trí đầu tiên (gốc) trong max h
}

// Loại bỏ phần tử lớn nhất
template<class T>
bool Heap<T>::pop() {
    if (count == 0) return false;  // Nếu heap rỗng, không thể xóa, trả về false

    // Thay thế gốc (phần tử lớn nhất) bằng phần tử cuối cùng
    elements[0] = elements[count - 1];
    count--;  // Giảm số lượng phần tử

    // Khôi phục tính chất của heap bằng cách di chuyển gốc mới xuống vị trí thích hợ
    reheapDown(0);
    return true;  // Xóa thành công
}

// Kiểm tra một phần tử có tồn tại trong heap không
template<class T>
bool Heap<T>::contains(T item) {
    // Duyệt qua tất cả các phần tử để tìm kiếm
    for (int i = 0; i < count; i++) {
        if (elements[i] == item) return true;  // Nếu tìm thấy, trả về true
    }
    return false;  // Không tìm thấy, trả về false
}

```

	Test	Expected	Got	

✓	Heap<int> maxHeap;
	for (int i=0;i<10;i++){
	maxHeap.push(i);
	}
	cout << maxHeap.size();
		10	10	✓

	Test	Expected	Got	

✓	`Heap<int> maxHeap;`
	`for (int i=0;i<10;i++){`
	`  maxHeap.push(i);`
	`}`
	`cout << maxHeap.isEmpty();`
		`0`	`0`	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 1,00

Implement function push to push a new item to a maxHeap. You also have to implement ensureCapacity and reheapUp to help you achieve that.

```
template
class Heap{
protected:
    T *elements;
    int capacity;
    int count;

public:
    Heap()
    {
        this->capacity = 10;
        this->count = 0;
        this->elements = new T[capacity];
    }
    ~Heap()
    {
        delete []elements;
    }
    void push(T item);
    void printHeap()
    {
        cout << "Max Heap [ ";
        for (int i = 0; i < count; i++)
            cout << elements[i] << " ";
        cout << "]";
    }

private:
    void ensureCapacity(int minCapacity);
    void reheapUp(int position);
};

// Your code here
```

For example:

Test	Result

`Heap<int> maxHeap;`
`for(int i = 0; i <5;i++)`
`  maxHeap.push(i);`
`maxHeap.printHeap();`
	`Max Heap [ 4 3 1 0 2 ]`

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
1  // Thêm một phần tử mới vào heap
2  template<class T>
3  void Heap<T>::push(T item) {
4      ensureCapacity(count + 1);  // Đảm bảo có đủ dung lượng
5      elements[count] = item;     // Thêm phần tử mới vào cuối
6      reheapUp(count);            // Điều chỉnh vị trí để duy trì tính chất max heap
7      count++;                    // Tăng số lượng phần tử
8  }
9
10 // Đảm bảo mảng có đủ dung lượng cho các phần tử mới
11 template<class T>
12 void Heap<T>::ensureCapacity(int minCapacity) {
13     if (minCapacity > capacity) {  // Nếu cần dung lượng lớn hơn hiện tại
14         int newCapacity = capacity * 2;  // Tăng gấp đôi dung lượng
15         if (newCapacity < minCapacity) newCapacity = minCapacity;  // Đảm bảo đủ dung
```

```
16
17          // Tạo mảng mới với dung lượng lớn hơn
18          T* newElements = new T[newCapacity];
19          // Sao chép các phần tử từ mảng cũ sang mảng mới
20          for (int i = 0; i < count; i++) {
21              newElements[i] = elements[i];
22          }
23
24          delete[] elements;  // Giải phóng bộ nhớ của mảng cũ
25          elements = newElements;  // Cập nhật con trỏ elements trỏ đến mảng mới
26          capacity = newCapacity;  // Cập nhật dung lượng mới
27      }
28  }
29
30  // Điều chỉnh heap sau khi thêm phần tử mới
31  template<class T>
32  void Heap<T>::reheapUp(int position) {
33      if (position <= 0) return;  // Nếu đã ở gốc, không cần điều chỉnh nữa
34
35      int parent = (position - 1) / 2;  // Tính vị trí của nút cha
36      // Nếu phần tử hiện tại lớn hơn phần tử cha (vi phạm tính chất max heap)
37      if (elements[position] > elements[parent]) {
38          // Hoán đổi phần tử hiện tại với phần tử cha
39          T temp = elements[position];
40          elements[position] = elements[parent];
41          elements[parent] = temp;
42          // Tiếp tục điều chỉnh từ vị trí cha
43          reheapUp(parent);
44      }
45  }
```

	Test	Expected	Got	

✓	`Heap<int> maxHeap;`
	`for(int i = 0; i <5;i++)`
	`  maxHeap.push(i);`
	`maxHeap.printHeap();`
		`Max Heap [ 4 3 1 0 2 ]`	`Max Heap [ 4 3 1 0 2 ]`	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 1,00

Given an array which the elements in it are random. Now we want to build a Max heap from this array. Implement functions Reheap up and Reheap down to heapify element at index position. We will use it to build a heap in next question.

To keep things simple, this question will separate the heap array, not store it in the class heap

```
void reheapDown(int maxHeap[], int numberOfElements, int index);
void reheapUp(int maxHeap[], int numberOfElements, int index);
```

For example:

Test	Result

int arr[] = {1,2,3,4,5,6,7,8};
int size = sizeof(arr)/sizeof(arr[0]);
reheapDown(arr,size,0);
cout << "[";
for(int i=0;i<size;i++)
  cout << arr[i] << " ";
cout << "]";
	[3 2 7 4 5 6 1 8]

int arr[] = {1,2,3,4,5,6,7,8};
int size = sizeof(arr)/sizeof(arr[0]);
reheapUp(arr,size,7);
cout << "[";
for(int i=0;i<size;i++)
  cout << arr[i] << " ";
cout << "]";
	[8 1 3 2 5 6 7 4]

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
// Hàm reheapDown cho một max heap
// Hàm này được sử dụng khi một phần tử có thể nhỏ hơn các phần tử con của nó
// và cần được di chuyển xuống để duy trì tính chất của max heap
void reheapDown(int maxHeap[], int numberOfElements, int index) {
    // Tính chỉ số của con trái và con phải
    int leftChildIndex = 2 * index + 1;
    int rightChildIndex = 2 * index + 2;
    int largestIndex = index; // Giả định vị trí hiện tại có giá trị lớn nhất

    // Kiểm tra nếu con trái tồn tại và có giá trị lớn hơn giá trị lớn nhất hiện tại
    if (leftChildIndex < numberOfElements && maxHeap[leftChildIndex] > maxHeap[larges
        largestIndex = leftChildIndex;
    }

    // Kiểm tra nếu con phải tồn tại và có giá trị lớn hơn giá trị lớn nhất hiện tại
    if (rightChildIndex < numberOfElements && maxHeap[rightChildIndex] > maxHeap[larg
        largestIndex = rightChildIndex;
    }

    // Nếu giá trị lớn nhất không nằm ở vị trí hiện tại, hoán đổi và tiếp tục reheap
    if (largestIndex != index) {
        // Hoán đổi các phần tử
        int temp = maxHeap[index];
        maxHeap[index] = maxHeap[largestIndex];
        maxHeap[largestIndex] = temp;

        // Đệ quy reheap down từ vị trí mới
        reheapDown(maxHeap, numberOfElements, largestIndex);
    }
}

// Hàm reheapUp cho một max heap
// Hàm này được sử dụng khi một phần tử có thể lớn hơn phần tử cha của nó
```

```cpp
// va can được di chuyen len de duy tri tinh chat cua max heap
void reheapUp(int maxHeap[], int numberOfElements, int index) {
    // Nếu đang ở gốc hoặc phần tử cha lớn hơn, không cần làm gì cả
    if (index == 0) {
        return;
    }

    // Tính chỉ số của phần tử cha
    int parentIndex = (index - 1) / 2;

    // Nếu phần tử hiện tại lớn hơn phần tử cha, hoán đổi chúng và tiếp tục reheap up
    if (maxHeap[index] > maxHeap[parentIndex]) {
        // Hoán đổi các phần tử
        int temp = maxHeap[index];
        maxHeap[index] = maxHeap[parentIndex];
        maxHeap[parentIndex] = temp;

        // Đệ quy reheap up từ vị trí cha
        reheapUp(maxHeap, numberOfElements, parentIndex);
    }
}
```

	Test	Expected	Got	

✓	`int arr[] = {1,2,3,4,5,6,7,8};`
	`int size = sizeof(arr)/sizeof(arr[0]);`
	`reheapDown(arr,size,0);`
	`cout << "[ ";`
	`for(int i=0;i<size;i++)`
	`  cout << arr[i] << " ";`
	`cout << "]";`
		`[ 3 2 7 4 5 6 1 8 ]`	`[ 3 2 7 4 5 6 1 8 ]`	✓

✓	`int arr[] = {1,2,3,4,5,6,7,8};`
	`int size = sizeof(arr)/sizeof(arr[0]);`
	`reheapUp(arr,size,7);`
	`cout << "[ ";`
	`for(int i=0;i<size;i++)`
	`  cout << arr[i] << " ";`
	`cout << "]";`
		`[ 8 1 3 2 5 6 7 4 ]`	`[ 8 1 3 2 5 6 7 4 ]`	✓

✓	`int arr[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};`
	`int size = sizeof(arr)/sizeof(arr[0]);`
	`reheapUp(arr,size,13);`
	`reheapUp(arr,size,12);`
	`cout << "[ ";`
	`for(int i=0;i<size;i++)`
	`  cout << arr[i] << " ";`
	`cout << "]";`
		`[ 14 2 13 4 5 1 3 8 9 10 11 12 6 7 15 ]`	`[ 14 2 13 4 5 1 3 8 9 10 11 12 6 7 15 ]`	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 1,00

Implement method remove to **remove** the element with given value from a **maxHeap**, **clear** to remove all elements and bring the heap back to the initial state. You also have to implement method **getItem** to help you. Some given methods that you don't need to implement again are **push**, **printHeap**, **ensureCapacity**, **reheapUp**, **reheapDown**.

```
class Heap {
protected:
    T* elements;
    int capacity;
    int count;
public:
    Heap()
    {
        this->capacity = 10;
        this->count = 0;
        this->elements = new T[capacity];
    }
    ~Heap()
    {
        delete[]elements;
    }
    void push(T item);
    int getItem(T item);
    void remove(T item);
    void clear();
    void printHeap()
    {
        cout << "Max Heap [ ";
        for (int i = 0; i < count; i++)
            cout << elements[i] << " ";
        cout << "]\n";
    }
private:
    void ensureCapacity(int minCapacity);
    void reheapUp(int position);
    void reheapDown(int position);
};

// Your code here
```

For example:

Test	Result

`Heap<int> maxHeap;`
`int arr[] = {42,35,30,15,20,21,18,3,7,14};`
`for (int i = 0; i < 10; i++)`
`  maxHeap.push(arr[i]);`
`maxHeap.remove(42);`
`maxHeap.remove(35);`
`maxHeap.remove(30);`
`maxHeap.printHeap();`
	`Max Heap [ 21 20 18 15 14 7 3 ]`

`Heap<int> maxHeap;`
`int arr[] = {78, 67, 32, 56, 8, 23, 19, 45};`
`for (int i = 0; i < 8; i++)`
`  maxHeap.push(arr[i]);`
`maxHeap.remove(78);`
`maxHeap.printHeap();`
	`Max Heap [ 67 56 32 45 8 23 19 ]`

Test	Result

Heap<int> maxHeap;
int arr[] = { 13, 19, 20, 7, 15, 12, 16, 10, 8, 9, 3, 6, 18, 2, 14, 1, 17, 4, 11, 5 };
for (int i = 0; i < 20; ++i)
maxHeap.push(arr[i]);
maxHeap.clear();
maxHeap.printHeap();
	Max Heap [ ]

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
template<class T>
int Heap<T>::getItem(T item) {
    // Tìm vị trí của phần tử có giá trị bằng item
    for (int i = 0; i < count; i++) {
        if (elements[i] == item) {
            return i; // Trả về chỉ số nếu tìm thấy
        }
    }
    return -1; // Trả về -1 nếu không tìm thấy phần tử
}

template<class T>
void Heap<T>::remove(T item) {
    // Tìm vị trí của phần tử cần xóa
    int index = getItem(item);

    // Nếu không tìm thấy phần tử, kết thúc
    if (index == -1) return;

    // Thay thế phần tử cần xóa bằng phần tử cuối cùng
    elements[index] = elements[count - 1];
    count--; // Giảm kích thước heap

    // Nếu heap rỗng hoặc phần tử vừa xóa là phần tử cuối cùng, không cần sắp xếp lại
    if (count <= 0 || index == count) return;

    // Điều chỉnh heap để duy trì tính chất max heap
    // Nếu phần tử mới lớn hơn phần tử cha, reheapUp
    if (index > 0 && elements[index] > elements[(index - 1) / 2]) {
        reheapUp(index);
    }
    // Ngược lại, reheapDown
    else {
        reheapDown(index);
    }
}

template<class T>
void Heap<T>::clear() {
    // Đặt lại số lượng phần tử về 0
    count = 0;
    // Không cần giải phóng bộ nhớ vì chúng ta vẫn giữ mảng với dung lượng hiện tại
    // Chỉ cần đặt lại số lượng phần tử, các phần tử cũ sẽ bị ghi đè khi thêm phần tử
}
```

	Test	Expected	Got	

✓	Heap<int> maxHeap;
	int arr[] = {42,35,30,15,20,21,18,3,7,14};
	for (int i = 0; i < 10; i++)
	maxHeap.push(arr[i]);
	maxHeap.remove(42);
	maxHeap.remove(35);
	maxHeap.remove(30);
	maxHeap.printHeap();
		Max Heap [21 20 18 15 14 7 3]	Max Heap [21 20 18 15 14 7 3]	✓

	Test	Expected	Got	

✓	`Heap<int> maxHeap;`
	`int arr[] = {78, 67, 32, 56, 8, 23, 19, 45};`
	`for (int i = 0; i < 8; i++)`
	`  maxHeap.push(arr[i]);`
	`maxHeap.remove(78);`
	`maxHeap.printHeap();`
		`Max Heap [ 67 56 32 45 8 23 19 ]`	`Max Heap [ 67 56 32 45 8 23 19 ]`	✓

✓	`Heap<int> maxHeap;`
	`int arr[] = { 13, 19, 20, 7, 15, 12, 16, 10, 8, 9, 3, 6, 18, 2, 14, 1, 17, 4, 11, 5 };`
	`for (int i = 0; i < 20; ++i)`
	`  maxHeap.push(arr[i]);`
	`maxHeap.clear();`
	`maxHeap.printHeap();`
		`Max Heap [ ]`	`Max Heap [ ]`	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **5**

Đúng

Đạt điểm 1,00 trên 1,00

Your task is to implement heap sort (in ascending order) on an unsorted array.

```
#define SEPARATOR "#<ab@17943918#@>#"
#ifndef SORTING_H
#define SORTING_H
#include <iostream>
#include <queue>
using namespace std;
template <class T>
class Sorting {
public:
    /* Function to print an array */
    static void printArray(T *start, T *end)
    {
        long size = end - start;
        for (int i = 0; i < size - 1; i++)
            cout << start[i] << ", ";
        cout << start[size - 1];
        cout << endl;
    }

    //Helping functions go here
    static void heapSort(T* start, T* end){
        //TODO
        Sorting<T>::printArray(start,end);
    }

};
#endif /* SORTING_H */
```

For example:

Test	Result
int arr[4]={4,2,9,1}; Sorting<int>::heapSort(&arr[0],&arr[4]);	1, 2, 4, 9
int arr[4]={-1,0,2,3}; Sorting<int>::heapSort(&arr[0],&arr[4]);	-1, 0, 2, 3

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
// Hàm heapify để chuyển một cây con có gốc tại vị trí i thành một max heap
    static void heapify(T* arr, int n, int i) {
        int largest = i;  // Khởi tạo largest là root
        int left = 2 * i + 1;  // Vị trí con trái = 2*i + 1
        int right = 2 * i + 2;  // Vị trí con phải = 2*i + 2

        // Nếu con trái lớn hơn gốc
        if (left < n && arr[left] > arr[largest])
            largest = left;

        // Nếu con phải lớn hơn gốc hoặc con trái
        if (right < n && arr[right] > arr[largest])
            largest = right;

        // Nếu largest không phải là gốc
        if (largest != i) {
            swap(arr[i], arr[largest]);

            // Đệ quy heapify cho cây con bị ảnh hưởng
            heapify(arr, n, largest);
        }
    }
```

```
//Helping functions go here
static void heapSort(T* start, T* end){
    int n = end - start;

        // Xây dựng max heap (sắp xếp mảng)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(start, n, i);

        // Trích xuất từng phần tử từ heap
        for (int i = n - 1; i > 0; i--) {
            // Di chuyển current root về cuối
            swap(start[0], start[i]);

            // Gọi max heapify cho heap đã giảm kích thước
            heapify(start, i, 0);
        }

    Sorting<T>::printArray(start,end);
}
```

	Test	Expected	Got	
✓	int arr[4]={4,2,9,1}; Sorting<int>::heapSort(&arr[0],&arr[4]);	1, 2, 4, 9	1, 2, 4, 9	✓
✓	int arr[4]={-1,0,2,3}; Sorting<int>::heapSort(&arr[0],&arr[4]);	-1, 0, 2, 3	-1, 0, 2, 3	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **6**

Đúng

Đạt điểm 1,00 trên 1,00

Given an array of non-negative integers. Each time, we can take the smallest integer out of the array, multiply it by 2, and push it back to the array.

Request: Implement function:

```
int leastAfter(vector<int>& nums, int k);
```

Where `nums` is the given array (the length of the array is between 1 and 100000). This function returns the smallest integer in the array after performing the operation `k` times (`k` is between 1 and 100000).

Example:

Given `nums = [2, 3, 5, 7]`.

In the 1st operation, we take `2` out and push back `4`. The array is now `nums = [3, 4, 5, 7]`.

In the 2nd operation, we take `3` out and push back `6`. The array is now `nums = [4, 5, 6, 7]`.

In the 3rd operation, we take `4` out and push back `8`. The array is now `nums = [5, 6, 7, 8]`.

With `k = 3`, the result would be `5`.

Note:

In this exercise, the libraries `iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm` has been included and `namespace std` are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

For example:

Test	Result

`vector<int> nums {2, 3, 5, 7};`
`int k = 3;`
`cout << leastAfter(nums, k);`
	5

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
// Hàm thực hiện k lần lấy số nhỏ nhất, nhân đôi và trả lại số nhỏ nhất cuối cùng
int leastAfter(vector<int>& nums, int k) {
    // Khởi tạo min-heap (priority_queue kiểu nhỏ nhất)
    priority_queue<int, vector<int>, greater<int>> minHeap;

    // Đưa tất cả phần tử mảng vào min-heap
    for (int num : nums) {
        minHeap.push(num);
    }

    // Thực hiện k lần thao tác: lấy nhỏ nhất, nhân đôi, đưa lại vào heap
    for (int i = 0; i < k; ++i) {
        int smallest = minHeap.top(); // lấy số nhỏ nhất
        minHeap.pop();                // loại bỏ khỏi heap
        minHeap.push(smallest * 2);   // nhân đôi và đưa lại vào heap
    }

    // Giá trị nhỏ nhất còn lại sau k lần thao tác
    return minHeap.top();
}
```

	Test	Expected	Got	

✓	`vector<int> nums {2, 3, 5, 7};`
	`int k = 3;`
	`cout << leastAfter(nums, k);`
		5	5	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **7**

Đúng

Đạt điểm 1,00 trên 1,00

Cho template của class PrinterQueue có 2 phương thức bắt buộc:

1. `addNewRequest(int priority, string fileName)`

Phương thức đầu tiên sẽ thêm 1 file vào danh sách hàng đợi của máy in (bao gồm độ ưu tiên và tên file). Test case sẽ có tối đa 100 file cùng lúc trong hàng đợi

2. `print()`

Phương thức thứ hai sẽ in tên file kèm xuống dòng và xóa nó ra khỏi hàng đợi. Nếu không có file nào trong hàng đợi, phương thức sẽ in ra "No file to print" kèm xuống dòng.

PrinterQueue tuân theo các quy tắc sau:

- fileName có độ ưu tiên cao nhất sẽ được in trước.
- Các fileName có cùng độ ưu tiên sẽ in theo thứ tự FIFO (First In First Out) order.

Nhiệm vụ của bạn là hiện thực class PrinterQueue thỏa mãn các yêu cầu dữ liệu trên

Lưu ý: Bạn có thể thay đổi mọi thứ, thêm thư viện cần thiết ngoại trừ thay đổi tên class, prototype của 2 public method bắt buộc.

Giải thích testcase 1: File goodbye.pdf có độ ưu tiên là 2 và được thêm vào sớm hơn file goodnight.pdf (độ ưu tiên = 2) nên sẽ được in trước, sau đó đến file goodnight.pdf và cuối cùng là hello.pdf có độ ưu tiên thấp nhất (1)

For example:

Test	Result

`PrinterQueue* myPrinterQueue = new PrinterQueue();`
`myPrinterQueue->addNewRequest(1, "hello.pdf");`
`myPrinterQueue->addNewRequest(2, "goodbye.pdf");`
`myPrinterQueue->addNewRequest(2, "goodnight.pdf");`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
	`goodbye.pdf`
	`goodnight.pdf`
	`hello.pdf`

`PrinterQueue* myPrinterQueue = new PrinterQueue();`
`myPrinterQueue->addNewRequest(1, "hello.pdf");`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
	`hello.pdf`
	`No file to print`
	`No file to print`

Answer: (penalty regime: 0, 0, 0, 100 %)

Reset answer

```
#include <iostream>
#include <string>
#include <queue>
#include <map>

using namespace std;

class PrinterQueue {
private:
    // map: key là priority, value là queue các file cùng priority đó
    map<int, queue<string>> requestMap;

public:
    // Thêm file mới vào hàng đợi
    void addNewRequest(int priority, string fileName) {
        requestMap[priority].push(fileName);
    }

    // In file theo quy tắc: ưu tiên cao nhất trước, cùng ưu tiên thì FIFO
    void print() {
        // Nếu hàng đợi trống
        if (requestMap.empty()) {
            cout << "No file to print" << endl;
```

```
            cout << "No file to print" << endl;
            return;
        }

        // Tìm priority cao nhất (map sắp xếp tăng dần => rbegin() là lớn nhất)
        auto it = requestMap.rbegin();
        int highestPriority = it->first;
        string fileToPrint = it->second.front();
        it->second.pop(); // Xóa file vừa in ra khỏi queue

        cout << fileToPrint << endl;

        // Nếu queue tại priority này đã hết file, xóa luôn priority đó khỏi map
        if (it->second.empty()) {
            requestMap.erase(highestPriority);
        }
    }
};
```

	Test	Expected	Got	

✓	`PrinterQueue* myPrinterQueue = new PrinterQueue();`
	`myPrinterQueue->addNewRequest(1, "hello.pdf");`
	`myPrinterQueue->addNewRequest(2, "goodbye.pdf");`
	`myPrinterQueue->addNewRequest(2, "goodnight.pdf");`
	`myPrinterQueue->print();`
	`myPrinterQueue->print();`
	`myPrinterQueue->print();`
		goodbye.pdf
		goodnight.pdf
		hello.pdf
			goodbye.pdf
			goodnight.pdf
			hello.pdf
				✓

✓	`PrinterQueue* myPrinterQueue = new PrinterQueue();`
	`myPrinterQueue->addNewRequest(1, "hello.pdf");`
	`myPrinterQueue->print();`
	`myPrinterQueue->print();`
	`myPrinterQueue->print();`
		hello.pdf
		No file to print
		No file to print
			hello.pdf
			No file to print
			No file to print
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.