Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Hai, 8 tháng 4 2024, 9:01 PM
Kết thúc lúc	Thứ Bảy, 13 tháng 4 2024, 3:38 PM
Thời gian thực hiện	4 Các ngày 18 giờ

Câu hỏi 1

Đúng

Đạt điểm 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`

`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

```
template<class T>
int Heap<T>::size(){
    return this->count;
}

template<class T>
bool Heap<T>::isEmpty(){
    return this->count == 0;
}

template<class T>
T Heap<T>::peek(){
    if (this->isEmpty()) return -1;
    return elements[0];
}

template<class T>
bool Heap<T>::contains(T item){
    for (int i = 0; i < size() - 1; i++){
        if (elements[i] == item){
            return true;
        }
    }
    return false;
}

template<class T>
bool Heap<T>::pop(){
    if (this->isEmpty()) return false;
    this->elements[0] = this->elements[this->size()-1];
```

```
31      this->elements[size()-1] = 0;
32      this->count--;
33      this->reheapDown(0);
34      return true;
35  }
```

	Test	Expected	Got	

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

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

Passed all tests! ✓

Câu hỏi **2**

Đúng

Đạt điểm 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

```
template<class T>
void Heap<T>::push(T item){
    ensureCapacity(count+1);
    elements[count] = item;
    count++;
    reheapUp(count-1);
}

template<class T>
void Heap<T>::ensureCapacity(int minCapacity){
    if (minCapacity > capacity){
    int newCapacity = max(capacity * 2, minCapacity);
    T* newElements = new T[newCapacity];
    for (int i = 0; i < count; i++) {
        newElements[i] = elements[i];
        }
    delete[] elements;
    elements = newElements;
    capacity = newCapacity;
    }
}

template<class T>
void Heap<T>::reheapUp(int position){
    int parentIndex = (position - 1)/2;
    if (position == 0 || elements[parentIndex] >= elements[position]) return;
    swap (elements[parentIndex], elements[position]);
    reheapUp(parentIndex);
}
```

	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! ✓

Câu hỏi **3**

Đúng

Đạt điểm 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

```
1  void reheapDown(int maxHeap[], int numberOfElements, int index)
2  {
3      int LargeIndex = index;
4      int LeftChildIndex = index*2 + 1;
5      int RightChildIndex = index*2 + 2;
6      if (LeftChildIndex < numberOfElements && maxHeap[LeftChildIndex] > maxHeap[LargeInde
7          LargeIndex = LeftChildIndex;
8      }
9      if (RightChildIndex < numberOfElements && maxHeap[RightChildIndex] > maxHeap[LargeIn
10         LargeIndex = RightChildIndex;
11     }
12     if (LargeIndex != index){
13         swap(maxHeap[LargeIndex], maxHeap[index]);
14         reheapDown(maxHeap, numberOfElements, LargeIndex);
15     }
```

```
}

void reheapUp(int maxHeap[], int numberOfElements, int index)
{
    int parentIndex = (index - 1) / 2;

    if (index > 0 && maxHeap[index] > maxHeap[parentIndex]) {
        swap(maxHeap[index], maxHeap[parentIndex]);
        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! ✓

Câu hỏi **4**

Đúng

Đạt điểm 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]

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: 10, 20, 30, ... %)

Reset answer

```
template<class T>
int Heap<T>::getItem(T item) {
    // TODO: return the index of item in heap
    for (int i = 0; i < count; i++){
        if (elements[i] == item){
            return i;
        }
    }
    return -1;
}

template<class T>
void Heap<T>::remove(T item) {
    // TODO: remove the element with value equal to item
    int IndexToRemove = getItem(item);
    if (IndexToRemove == -1) return;
    elements[IndexToRemove] = elements[--count];
    if (IndexToRemove > 0 && elements[IndexToRemove] > elements[(IndexToRemove - 1)/2]){
        reheapUp(IndexToRemove);
    } else {
        reheapDown(IndexToRemove);
    }
```

```
}

template<class T>
void Heap<T>::clear() {
    // TODO: delete all elements in heap
    //    then reallocate memory as initial state
    delete[] elements;
    this->capacity = 10;
    this->count = 0;
    elements = new T[capacity];
}
```

	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 ]`	✓

✓	`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! ✓

Câu hỏi **5**

Đúng

Đạt điểm 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

```
1  //Helping functions go here
2  static void heapify (int maxHeap[], int numberOfElements, int index){
3      int LargeIndex = index;
4      int LeftChildIndex = index*2 + 1;
```

```
    int RightChildIndex = index*2 + 2;
    if (LeftChildIndex < numberOfElements && maxHeap[LeftChildIndex] > maxHeap[LargeInde
        LargeIndex = LeftChildIndex;
    }
    if (RightChildIndex < numberOfElements && maxHeap[RightChildIndex] > maxHeap[LargeIn
        LargeIndex = RightChildIndex;
    }
    if (LargeIndex != index){
        swap(maxHeap[LargeIndex], maxHeap[index]);
        heapify(maxHeap,numberOfElements,LargeIndex);
    }
}
static void heapSort(T* start, T* end){
    //TODO
    int n  = end - start;
    for (int i = n/2 - 1; i >= 0; i--){
        heapify(start,n,i);
    }
    for (int i = n - 1; i > 0; i--){
        swap(start[0], start[i]);
        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	✓

	Test	Expected	Got	
✓	`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! ✓

Câu hỏi **6**

Đúng

Đạt điểm 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

```
1 ▾ class PrinterQueue {
2   private:
3 ▾     struct PrintRequest {
```

```
        int priority;
        string fileName;
        PrintRequest* next;

        PrintRequest(int p, const string& name) : priority(p), fileName(name), next(null
    };

    PrintRequest* head;

public:
    PrinterQueue() : head(nullptr) {}

    ~PrinterQueue() {
        while (head) {
            PrintRequest* temp = head;
            head = head->next;
            delete temp;
        }
    }

    void addNewRequest(int priority, const string& fileName) {
        PrintRequest* newRequest = new PrintRequest(priority, fileName);
        if (!head || priority > head->priority) {
            newRequest->next = head;
            head = newRequest;
        } else {
            PrintRequest* temp = head;
            while (temp->next && temp->next->priority >= priority) {
                temp = temp->next;
            }
            newRequest->next = temp->next;
            temp->next = newRequest;
        }
    }

    void print() {
        if (!head) {
            cout << "No file to print" << endl;
            return;
        }

        PrintRequest* current = head;
        cout << current->fileName << endl;
        head = head->next;
        delete current;
    }
};
```

	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! ✓

Câu hỏi **7**

Đúng

Đạt điểm 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

```
int leastAfter(vector<int>& nums, int k) {
    priority_queue<int, vector<int>, greater<int>> pq(nums.begin(), nums.end());
    while (k--) {
        int smallest = pq.top();
        pq.pop();
        pq.push(smallest * 2);
    }
    return pq.top();
}
```

	Test	Expected	Got	

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

Passed all tests! ✓