$\underline{\text{Github:}}\ \text{LoreDN}$

Author: Lorenzo Di Napoli

 $Repository: \ https://github.com/LoreDN/code-Cpp$

myHeap — Documentation

Contents

Library Description	2
CONTENTS OF THE LIBRARY	2
INHERITARY STRUCTURE	2
TEMPLATES IMPLEMENTATION	2
Classes Documentation	3
ROOT. myHeap	4
1. myHeap_Min	7
1. myHeap_Max	8

Library Description

This library contains the definition of the "myHeap" class, which allows to works with a set implementation of an Heap in C++.

Since an Heap can be implemented as a Min-Heap or a Max-Heap, there are two implementations, the "myHeap Min" one and the "myHeap Max" one.

CONTENTS OF THE LIBRARY

The library consists of two files:

- "myHeap.hpp": the template library, it is the header file where you can find a brief description on how to use the library in practice, together with all the classes definitions.
- "myHeap.cpp": the core of the library, it contains all the methods definitions for each class; it is the file that has to be given to the compiler in order to be able to use the library.

INHERITARY STRUCTURE

The library starts with the definition of a simple abstract class: "myHeap".

Starting from this class, there are two derivated classes, "myHeap_Min" and "myHeap_Max", which allow to use the core class as a Min-Heap or a Max-Heap, implementing both the solutions.

TEMPLATES IMPLEMENTATION

In order to use the Heaps with all possibles types and custom classes, have been used templates.

Due to the library structure, the correct way to use them is via **explicit template instantiations**, which have to be manually added for each wanted custom type/class.

This process has been made as easy as possible; the only thing you need to do is to add the wanted **explicit template instantiation** at the start of the "myHeap.cpp" file. Have been already added the *int* and *float* types ones, so you can copy one of these and change the type with the wanted one.

Classes Documentation

In the "myHeap" library have been implemented a total of three classes:

- R. myHeap: the Root abstract class, from which all the others derive from.
- 1. $myHeap_Min$: Heap implementation as a Min-Heap.
- 2. $\mathbf{myHeap_Max:}$ Heap implementation as a Max-Heap.

ROOT. myHeap

```
template <typename TYPE>
class myHeap {
    protected:
         // attributes
         size_t size;
         size_t usage;
         TYPE *heap;
         // methods prototypes
         void resize();
         void switch_keys(const size_t first, const size_t second);
         virtual void heapify_up(int index) = 0;
         virtual void heapify_down(size_t index) = 0;
    public:
         // constructor
         myHeap(const size_t dim);
         // destructor
         virtual ~myHeap() = default;
         // methods prototypes
         inline size_t get_size() = { return this->size; };
         inline size_t get_usage() = { return this->usage; };
         inline TYPE get_first() { return this->heap[0]; };
         inline bool is_empty() { return !(this->usage); };
         void scan(const size_t num);
         void scan_file(const std::string path);
         void print(const size_t start, const size_t end, const bool
             flag_user_interface);
         void print_file(const std::string path, const size_t start, const size_t end,
             const bool flag_user_interface);
         void push(const TYPE key);
         TYPE pop();
};
```

size_t size;

The Heap size, it is a *protected* member, since the user is not able to modify it.

size_t usage;

The number of elements currently stored in the Heap, it is a *protected* member, since the user is not able to modify it.

TYPE *heap;

The Heap itself, it is a *protected* member, since the user is not able to modify it directly.

void resize();

Method to resize the Heap, it is a protected member, since the user is not able to call it directly.

```
void switch_keys(const size_t first, const size_t second);
```

Method to switch two keys in the Heap, it is a *protected* member, since the user is not able to call it directly.

```
virtual void heapify_up(int index) = 0;
```

Method to heapify the Heap from bottom to top, it has been left as a *virtual method* since each derivated class has a different heapify-rule.

```
virtual void heapify_down(size_t index) = 0;
```

Method to heapify the Heap from top to bottom, it has been left as a *virtual method* since each derivated class has a different heapify-rule.

```
myHeap(const size_t dim);
```

This is the *constructor*, which assignes a size variable dim to the protected member myHeap.size. It also sets myHeap.usage to 0 and allocates memory for the Heap myHeap.heap.

```
virtual ~myHeap() = default;
```

This is the *destructor*, it has been left as a *virtual method* in order to allow each derivated class to implement its own one.

```
inline size_t get_size() = { return this->size; };
```

Getter Method for the Heap-size.

```
inline size_t get_usage() = { return this->usage; };
```

Getter Method for the Heap-usage.

```
inline TYPE get_first() { return this->heap[0]; };
```

Getter Method for the first element of the Heap.

```
inline bool is_empty() { return !(this->usage); };
```

Method to check if the Heap is empty.

```
void scan(const size_t num);
```

Method to scan a total of num elements from terminal.

```
void scan_file(const std::string path);
```

Method to scan all the elements from a file via std::ifstream.

```
void print(const size_t start, const size_t end, const bool flag_user_interface);
```

Method to print Heap to the terminal.

The const bool flag_user_interface is used in order to choose the type of print:

```
- flag = true \longrightarrow Bucket i: myHeap.heap[ i ] "\n".
```

```
- flag = false \longrightarrow myHeap.heap[i] "\n" myHeap.heap[i + 1] ...
```

```
void print_file(const std::string path, const size_t start, const size_t end, const
bool flag_user_interface);
```

Method to print Heap to file via std::ofstream.

The const bool flag_user_interface is used in order to choose the type of print:

```
- flag = true \longrightarrow Bucket i: myHeap.heap[i] "\n".
```

```
- flag = false -> myHeap.heap[ i ] "\n" myHeap.heap[ i + 1 ] ....
```

```
void push(const TYPE key);
```

Method to push a key to the Heap.

TYPE pop();

Method to pop the first key from the Heap.

1. myHeap_Min

```
void heapify_up(int index) override;
```

Method to heapify the Min-Heap from bottom to top.

```
void heapify_down(size_t index) override;
```

Method to heapify the Min-Heap from top to bottom.

```
myHeap_Min(const size_t dim);
```

This is the constructor, which calls the constructor of the Root class myHeap.

```
~myHeap_Min() override;
```

This is the destructor, which frees the Min-Heap $\verb|myHeap.heap|$.

1. myHeap_Max

```
void heapify_up(int index) override;
```

Method to heapify the Max-Heap from bottom to top.

```
void heapify_down(size_t index) override;
```

Method to heapify the Max-Heap from top to bottom.

```
myHeap_Max(const size_t dim);
```

This is the constructor, which calls the constructor of the Root class myHeap.

```
~myHeap_Max() override;
```

This is the destructor, which frees the Max-Heap $\verb|myHeap.heap|$.