1 Introduction

This is an exercise in class template design. This exercise covers the following topics:

- class templates
- nontype template parameters
- `static_assert`
- member function templates
- class template design
- asserts (`#include <cassert>`) (optional, but recommended)
- exceptions

2 Queue

In this exercise you will create a class that represents a queue. There are essentially two types of queues:

- A *dynamic queue* is a queue that can vary in size, i.e. you can enqueue as many values as you want!
- A *static queue* is a queue where there is a max capacity, i.e. there is a maximum amount of values that can be stored in the queue.

There are pros and cons of each implementation; a dynamic queue requires dynamic memory management (`new` and `delete`) which can be quite slow. But it is useful if we don't know how many values will be stored in our queue.

A static queue on the other hand allows us to place the queue directly on the stack (i.e. letting the compiler handle the memory), which is a lot faster than dynamic memory management. This is the prefered way to implement a queue if you know ahead of time how many values will be stored in the queue at any given time.

For this exercise we will focus on a static queue.

3 The exercise

In this exercise you are going to create a class template called `Queue` that takes two template parameters; `T`, the type of the values stored in the queue, and `N`,the max capacity of the queue. `Queue` should have the following member functions:

- `void enqueue(T value)`: inserts `value` at the end of the queue.
- `T dequeue()`: removes the first value from the front of the queue and return it.
- `bool empty()`: returns whether the queue is empty or not.

- `bool full()`: returns whether the max number of values has been inserted into the queue or not.
- `void clear()`: resets the queue to become empty.
- `T& front()`: returns a reference to the first element in the front of the queue.

It should also have a member function template called `copy_and_expand` that creates a copy of the queue and adds more capacity to that copy. This is done by taking a template parameter `M` that represents how many slots should be added to the capacity of the new queue.

You should also create an exception class called `queue_error` that is a derived class of `std::runtime_error` (defined in `<stdexcept>`).

`queue_error` should be thrown if the user tries to:

- enqueue values into an already full queue,
- dequeue a value from an empty queue,
- or access the front of an empty queue.

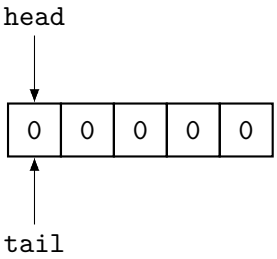
Also, you should make a custom compile error for when `N <= 0` telling the user that a queue must have space for at least `1` value.

There are some testcases given in `main.cc`.

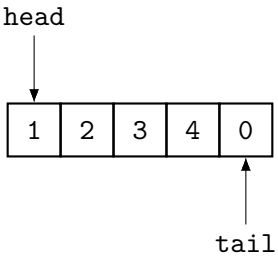
4 Implementation

The way a static queue can be implemented is by representing it as an array of the appropriate size and the keeping track of the beginning (head) and the end (tail) of the queue.

Suppose that we have an array of size 5. When it is empty it should look like this:

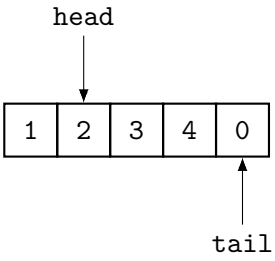


Now, suppose that we enqueue 4 elements; `1`, `2`, `3` and `4`, then it should look like this:

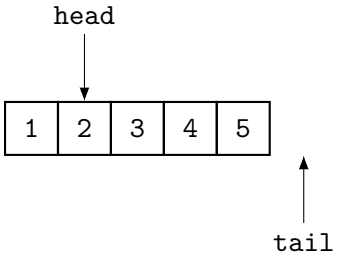


Notice that `tail` represents the first free slot in the array.

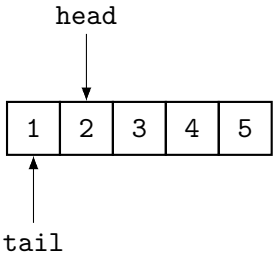
Now suppose that we dequeue the first element (the `1`), then the queue will look like this:



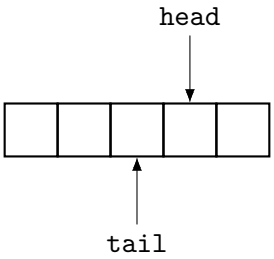
Let's enqueue the value 5:



Notice that the queue isn't full since there is one slot left at the front of the queue. Now we have two alternatives, either we let `tail` wrap-around to the beginning of the queue, like this:



OR we let `head` and `tail` keep on growing outside the bounds of the array indefinietly, and simply calculate their corresponding index in the array. For example: if `head` is `13` and `tail` is `17`, then we use modulus to get corresponding index: `13 % 5 == 3` and `17 % 5 == 2`. I.e. their indices correspond to this image:



There are different pros and cons to these approaches, but I won't tell them to you. Instead I want you to think of how the implementation of `Queue` differs depending on which approach you choose and pick the one you feel is the best.