Q.A:

Defining the normal iterator operations as const would not be sufficient to allow iteration on a fixed queue, because a const iterator is more restrictive than a normal iterator that has its operations defined as const.

For example, a normal iterator that has its operations defined as const is still capable of modifying the elements in the container, even though the operations themselves are const. This is because the const qualifier only applies to the operations and not to the iterator itself.

Q.B:

In general, in our implementation of "Queue" we had 3 assumptions for the type <T>, which that T has a **copy constructor**, a **destructor** and an **assignment operator**.

We made these assumptions in the following functions:

- **The pushBack() function** assumes that T has a copy constructor, so that it can create a copy of the item parameter and insert it into the queue.
- **The front() function** assumes that T has an assignment operator, so that it can assign the value of the front item in the queue to the return value.
- **The popFront** function assumes that T has a destructor.

Q.C:

The student will receive a linking error (for example `"undefined reference to` `` `Queue<T>::pushBack()'" ``), this occurs in the linking stage, because the compiler does not generate any code for the member functions of the class. Instead, it generates a template for the member functions that can be instantiated when the class is used. This means that the implementation of the member functions must be included in the source file where the class is defined, so that the compiler can generate the necessary code.

סעיף ד' בעמוד הבא

Q.D:

We can use Functor and add a field in the class for the number that she'll get in the running time and based on that number we'll use a suitable pre-written constructor for the functor.

this is an example for how to do it:

```cpp
class DivisibleBy
{
public:
    explicit DivisibleBy(int divisor) : m_divisor(divisor) {}

    bool operator()(int x) const { return x % m_divisor == 0; }

private:
    int m_divisor;
};

// ...

Queue<int> q;
for (int i = 1; i < 10; i++)
{
    q.pushBack(i);
}

int divisor = 3;

/**

* here we choose what number we want to use as a divisor (we can use another ways

* to get it - like "std::cin" for example)

*/

Queue<int> result = filter(q, DivisibleBy(divisor));
```