CPP Module 08

Ex00:

A first easy exercise is the way to start off on the right foot.
Write a function template easyfind that accepts a type T. It takes two parameters.
The first one has type T and the second one is an integer.
Assuming T is a container of integers, this function has to find the first occurrence of the second parameter in the first parameter.
If no occurrence is found, you can either throw an exception or return an error value of your choice. If you need some inspiration, analyze how standard containers behave.
Of course, implement and turn in your own tests to ensure everything works as expected.

Ex01:

Develop a Span class that can store a maximum of N integers. N is an unsigned int variable and will be the only parameter passed to the constructor.
This class will have a member function called addNumber() to add a single number to the Span. It will be used in order to fill it. Any attempt to add a new element if there are already N elements stored should throw an exception.
Next, implement two member functions: shortestSpan() and longestSpan()
They will respectively find out the shortest span or the longest span (or distance, if you prefer) between all the numbers stored, and return it. If there are no numbers stored, or only one, no span can be found. Thus, throw an exception.
Of course, you will write your own tests and they will be way more thorough than the ones below. Test your Span at least with a minimum of 10 000 numbers. More would be even better.
Last but not least, it would be wonderful to fill your Span using a range of iterators.
Making thousands calls to addNumber() is so annoying. Implement a member function to add many numbers to your Span in one call.

Ex02:

Now, time to move on more serious things. Let’s develop something weird.
The std::stack container is very nice. Unfortunately, it is one of the only STL Containers that is NOT iterable. That’s too bad.
But why would we accept this? Especially if we can take the liberty of butchering the original stack to create missing features.
To repair this injustice, you have to make the std::stack container iterable.
Write a MutantStack class. It will be implemented in terms of a std::stack.
It will offer all its member functions, plus an additional feature: iterators.
Of course, you will write and turn in your own tests to ensure everything works as expected.
If you run it a first time with your MutantStack, and a second time replacing the MutantStack with, for example, a std::list, the two outputs should be the same. Of course, when testing another container, update the code below with the corresponding member functions (push() can become push_back()).