


# Chapter V

## Exercise 01: Span

	Exercise : 01
Span	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <code>Makefile</code> , <code>main.cpp</code> , <code>Span.{h, hpp}</code> , <code>Span.cpp</code>	
Forbidden functions : None	

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.

Running this code:

```
int main()
{
    Span sp = Span(5);

    sp.addNumber(6);
    sp.addNumber(3);
    sp.addNumber(17);
    sp.addNumber(9);
    sp.addNumber(11);

    std::cout << sp.shortestSpan() << std::endl;
    std::cout << sp.longestSpan() << std::endl;

    return 0;
}
```

Should output:

```
$> ./ex01
2
14
$>
```

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.



If you don't have a clue, study the Containers. Some member functions take a range of iterators in order to add a sequence of elements to the container.