Size versus Capacity

We'll test and alter the capacity of a string.

The number of elements a string has (`str.size()`) is in general smaller than the number of elements for which space is reserved: `str.capacity()`. Therefore if we add elements to a string, new memory will not necessarily be allocated. `std:max_size()` return the maximum amount of elements a string can have. For the three methods the following relation holds: `str.size() <= str.capacity() <= str.max_size()`.

The following table shows the methods for dealing with memory management of strings.

Methods	Description
`str.empty()`	Checks if `str` has elements.
`str.size(), str.length()`	Number of elements of the `str`.
`str.capacity()`	Number of elements `str` can have without reallocation.
`str.max_size()`	Number of elements `str` can maximal have.
`str.resize(n)`	Increases `str` to n elements.
`str.reserve(n)`	Reserves memory for a least `n` elements.
`str.shrink_to_fit()`	Adjusts the capacity of the string to it's size.

	it's size.

The request `str.shrink_to_fit()` is, as in the case of std::vector, non-binding.

```
#include <iostream>
#include <string>

void showStringInfo(const std::string& s){

  std::cout << s << std::endl;
  std::cout << "s.size(): " << s.size() << std::endl;
  std::cout << "s.capacity(): " << s.capacity() << std::endl;
  std::cout << "s.max_size(): " << s.max_size() << std::endl;
  std::cout << std::endl;

}

int main(){

  std::string str;
  showStringInfo(str);

  str +="12345";
  showStringInfo(str);

  str.resize(30);
  showStringInfo(str);

  str.reserve(1000);
  showStringInfo(str);

  str.shrink_to_fit();
  showStringInfo(str);

}
```

Size versus capacity

In the next lesson, we'll learn how we can merge and compare strings.