

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 you add elements to a string, new memory may not be allocated automatically. `std::max_size()` returns the maximum number 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 the memory management of the string.

Methods	Description
<code>str.empty()</code>	Checks if <code>str</code> has elements.
<code>str.size()</code> , <code>str.length()</code>	Number of elements of the <code>str</code> .
<code>str.capacity()</code>	Number of elements <code>str</code> can have without reallocation.
<code>str.max_size()</code>	Number of elements <code>str</code> can maximal have.
<code>str.resize(n)</code>	Increases <code>str</code> to n elements.
<code>str.reserve(n)</code>	Reserve memory for a least <code>n</code> elements.
<code>str.shrink_to_fit()</code>	Adjusts the capacity of the string to its size

its 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