Modifying operations

There are a few modifying operations that a string view can perform. Some of these are unique to string views only. Let's check them out.

The call `stringView.swap(stringView2)` swaps the content of the two string views. The methods `remove_prefix` and `remove_suffix` are unique to a string view because a string supports neither. `remove_prefix` shrinks its start forward; `remove_suffix` shrinks its end backwards.

```
// string_view.cpp

#include <iostream>
#include <string>
#include <experimental/string_view>

int main(){

  std::string str = "   A lot of space";
  std::experimental::string_view strView = str;
  strView.remove_prefix(std::min(strView.find_first_not_of(" "), strView.size()));
  std::cout << "str      :  " << str << std::endl
            << "strView  : " << strView << std::endl;

  std::cout << std::endl;

  char arr[] = {'A', ' ', 'l', 'o', 't', ' ', 'o', 'f', ' ', 's', 'p', 'a', 'c', 'e', '\0',
  std::experimental::string_view strView2(arr, sizeof arr);
  auto trimPos = strView2.find('\0');
  if(trimPos != strView2.npos) strView2.remove_suffix(strView2.size() - trimPos);
  std::cout << "arr     : " << arr << ", size=" << sizeof arr << std::endl
            << "strView2: " << strView2 << ", size=" << strView2.size() << std::endl;

}
```

Non-modifying operations

> **No memory allocation with a string view**
> If you create a string view or copy a string view, there is no memory allocation necessary. This is in contrast to a string; creating a string or

> copying a string requires memory allocation.

```
// stringView.cpp
#include <cassert>
#include <iostream>
#include <string>

#include <string_view>

void* operator new(std::size_t count){
    std::cout << "   " << count << " bytes" << std::endl;
    return malloc(count);
}

void getString(const std::string& str){}

void getStringView(std::string_view strView){}

int main() {

    std::cout << std::endl;

    std::cout << "std::string" << std::endl;

    std::string large = "0123456789-123456789-123456789-123456789";
    std::string substr = large.substr(10);

    std::cout << std::endl;

    std::cout << "std::string_view" << std::endl;

    std::string_view largeStringView{large.c_str(), large.size()};
    largeStringView.remove_prefix(10);

    assert(substr == largeStringView);

    std::cout << std::endl;

    std::cout << "getString" << std::endl;

    getString(large);
    getString("0123456789-123456789-123456789-123456789");
    const char message []= "0123456789-123456789-123456789-123456789";
    getString(message);

    std::cout << std::endl;

    std::cout << "getStringView" << std::endl;

    getStringView(large);
    getStringView("0123456789-123456789-123456789-123456789");
    getStringView(message);

    std::cout << std::endl;

}
```

> 🔍 **Memory allocation**
> Thanks to the global overload `operator new` I can observe each memory allocation.