Chapter 5. Boost.StringAlgorithms

The Boost.StringAlgorithms library provides many free-standing functions for string manipulation. Strings can be of type `std::string`, `std::wstring`, or any other instance of the class template `std::basic_string`. This includes the string classes `std::u16string` and `std::u32string` introduced with C++11.

The functions are categorized within different header files. For example, functions converting from uppercase to lowercase are defined in `boost/algorithm/string/case_conv.hpp`. Because Boost.StringAlgorithms consists of more than 20 different categories and as many header files, `boost/algorithm/string.hpp` acts as the common header including all other header files for convenience.

Example 5.1. Converting strings to uppercase

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ Libraries";
  std::cout << to_upper_copy(s) << '\n';
}
```

The function `boost::algorithm::to_upper_copy()` converts a string to uppercase, and `boost::algorithm::to_lower_copy()` converts a string to lowercase. Both functions return a copy of the input string, converted to the specified case. To convert the string in place, use the functions `boost::algorithm::to_upper()` or `boost::algorithm::to_lower()`.

Example 5.1 converts the string “Boost C++ Libraries” to uppercase using `boost::algorithm::to_upper_copy()`. The example writes `BOOST C++ LIBRARIES` to standard output.

Functions from Boost.StringAlgorithms consider locales. Functions like `boost::algorithm::to_upper_copy()` use the global locale if no locale is passed explicitly as a parameter.

Example 5.2. Converting a string to uppercase with a locale

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <locale>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ k\xfct\xfcphaneleri";
  std::string upper_case1 = to_upper_copy(s);
  std::string upper_case2 = to_upper_copy(s, std::locale{"Turkish"});
  std::locale::global(std::locale{"Turkish"});
  std::cout << upper_case1 << '\n';
  std::cout << upper_case2 << '\n';
}
```

Example 5.2 calls `boost::algorithm::to_upper_copy()` twice to convert the Turkish string “Boost C++ kütüphaneleri” to uppercase. The first call to `boost::algorithm::to_upper_copy()` uses the global locale, which in this case is the C locale. In the C locale, there is no uppercase mapping for characters with umlauts, so the output will look like this: `BOOST C++ KüTüPHANELERI`.

The Turkish locale is passed to the second call to `boost::algorithm::to_upper_copy()`. Since this locale does have uppercase equivalents for umlauts, the entire string can be converted to uppercase. Therefore, the second call to `boost::algorithm::to_upper_copy()` correctly converts the string, which looks like this: `BOOST C++ KÜTÜPHANELERI`.

Note

If you want to run the example on a POSIX operating system, replace “Turkish” with “tr_TR”, and make sure the Turkish locale is installed.

Example 5.3. Algorithms to remove characters from a string

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ Libraries";
  std::cout << erase_first_copy(s, "s") << '\n';
  std::cout << erase_nth_copy(s, "s", 0) << '\n';
  std::cout << erase_last_copy(s, "s") << '\n';
  std::cout << erase_all_copy(s, "s") << '\n';
  std::cout << erase_head_copy(s, 5) << '\n';
  std::cout << erase_tail_copy(s, 9) << '\n';
}
```

Boost.StringAlgorithms provides several functions you can use to delete individual characters from a string (see Example 5.3). For example, `boost::algorithm::erase_all_copy()` will remove all occurrences of a particular character from a string. To remove only the first occurrence of the character, use `boost::algorithm::erase_first_copy()` instead. To shorten a string by a specific number of characters on either end, use the functions `boost::algorithm::erase_head_copy()` and `boost::algorithm::erase_tail_copy()`.

Example 5.4. Searching for substrings with `boost::algorithm::find_first()`

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ Libraries";
  boost::iterator_range<std::string::iterator> r = find_first(s, "C++");
  std::cout << r << '\n';
  r = find_first(s, "xyz");
  std::cout << r << '\n';
}
```

Functions such as `boost::algorithm::find_first()`, `boost::algorithm::find_last()`, `boost::algorithm::find_nth()`, `boost::algorithm::find_head()` and `boost::algorithm::find_tail()` are available to find strings within strings.

All of these functions return a pair of iterators of type `boost::iterator_range`. This class originates from Boost.Range, which implements a range concept based on the iterator concept. Because the operator `operator<<` is overloaded for `boost::iterator_range`, the result of the individual search algorithm can be written directly to standard output. Example 5.4 prints `C++` for the first result and an empty string for the second one.

Example 5.5. Concatenating strings with `boost::algorithm::join()`

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <vector>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::vector<std::string> v{"Boost", "C++", "Libraries"};
  std::cout << join(v, " ") << '\n';
}
```

A container of strings is passed as the first parameter to the function `boost::algorithm::join()`, which concatenates them separated by the second parameter. Example 5.5 will output `Boost C++ Libraries`.

Example 5.6. Algorithms to replace characters in a string

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ Libraries";
  std::cout << replace_first_copy(s, "+", "-") << '\n';
  std::cout << replace_nth_copy(s, "+", 0, "-") << '\n';
  std::cout << replace_last_copy(s, "+", "-") << '\n';
  std::cout << replace_all_copy(s, "+", "-") << '\n';
  std::cout << replace_head_copy(s, 5, "BOOST") << '\n';
  std::cout << replace_tail_copy(s, 9, "LIBRARIES") << '\n';
}
```

Like the functions for searching strings or removing characters from strings, Boost.StringAlgorithms also provides functions for replacing substrings within a string. These include the following functions: `boost::algorithm::replace_first_copy()`, boost::algorithm::replace_nth_copy(), boost::algorithm::replace_last_copy(), boost::algorithm::replace_all_copy(), boost::algorithm::replace_head_copy() and boost::algorithm::replace_tail_copy(). They can be applied in the same way as the functions for searching and removing, except they require an additional parameter – the replacement string (see Example 5.6).

Example 5.7. Algorithms to trim strings

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "\t Boost C++ Libraries \t";
  std::cout << "_" << trim_left_copy(s) << "_\n";
  std::cout << "_" << trim_right_copy(s) << "_\n";
  std::cout << "_" << trim_copy(s) << "_\n";
}
```

To remove spaces on either end of a string, use boost::algorithm::trim_left_copy(), boost::algorithm::trim_right_copy() and boost::algorithm::trim_copy() (see Example 5.7). The global locale determines which characters are considered to be spaces.

Boost.StringAlgorithms lets you provide a predicate as an additional parameter for different functions to determine which characters of the string the function is applied to. The versions with predicates are: boost::algorithm::trim_right_copy_if(), boost::algorithm::trim_left_copy_if(), and boost::algorithm::trim_copy_if().

Example 5.8. Creating predicates with boost::algorithm::is_any_of()

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "--Boost C++ Libraries--";
  std::cout << trim_left_copy_if(s, is_any_of("-")) << '\n';
  std::cout << trim_right_copy_if(s, is_any_of("-")) << '\n';
  std::cout << trim_copy_if(s, is_any_of("-")) << '\n';
}
```

Example 5.8 uses another function called boost::algorithm::is_any_of(), which is a helper function to create a predicate that checks whether a certain character – passed as parameter to is_any_of() – exists in a string. With boost::algorithm::is_any_of(), the characters for trimming a string can be specified. Example 5.8 uses the hyphen character.

Boost.StringAlgorithms provides many helper functions that return commonly used predicates.

Example 5.9. Creating predicates with boost::algorithm::is_digit()

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "123456789Boost C++ Libraries123456789";
  std::cout << trim_left_copy_if(s, is_digit()) << '\n';
  std::cout << trim_right_copy_if(s, is_digit()) << '\n';
  std::cout << trim_copy_if(s, is_digit()) << '\n';
}
```

The predicate returned by `boost::algorithm::is_digit()` tests whether a character is numeric. In Example 5.9, `boost::algorithm::is_digit()` is used to remove digits from the string **s**.

Boost.StringAlgorithms also provides helper functions to check whether a character is uppercase or lowercase: `boost::algorithm::is_upper()` and `boost::algorithm::is_lower()`. All of these functions use the global locale by default, unless you pass in a different locale as a parameter.

Besides the predicates that verify individual characters of a string, Boost.StringAlgorithms also offers functions that work with strings instead (see Example 5.10).

Example 5.10. Algorithms to compare strings with others

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ Libraries";
  std::cout.setf(std::ios::boolalpha);
  std::cout << starts_with(s, "Boost") << '\n';
  std::cout << ends_with(s, "Libraries") << '\n';
  std::cout << contains(s, "C++") << '\n';
  std::cout << lexicographical_compare(s, "Boost") << '\n';
}
```

The `boost::algorithm::starts_with()`, `boost::algorithm::ends_with()`, `boost::algorithm::contains()`, and `boost::algorithm::lexicographical_compare()` functions compare two individual strings.

Example 5.11 introduces a function that splits a string into smaller parts.

Example 5.11. Splitting strings with `boost::algorithm::split()`

```
#include <boost/algorithm/string.hpp>
#include <string>
#include <vector>
#include <iostream>

using namespace boost::algorithm;

int main()
```

```
{
  std::string s = "Boost C++ Libraries";
  std::vector<std::string> v;
  split(v, s, is_space());
  std::cout << v.size() << '\n';
}
```

With `boost::algorithm::split()`, a given string can be split based on a delimiter. The substrings are stored in a container. The function requires as its third parameter a predicate that tests each character and checks whether the string should be split at the given position. Example 5.11 uses the helper function `boost::algorithm::is_space()` to create a predicate that splits the string at every space character.

Many of the functions introduced in this chapter have versions that ignore the case of the string. These versions typically have the same name, except for a leading “i”. For example, the equivalent function to `boost::algorithm::erase_all_copy()` is `boost::algorithm::ierase_all_copy()`.

Finally, many functions of Boost.StringAlgorithms also support regular expressions. Example 5.12 uses the function `boost::algorithm::find_regex()` to search for a regular expression.

Example 5.12. Searching strings with `boost::algorithm::find_regex()`

```
#include <boost/algorithm/string.hpp>
#include <boost/algorithm/string/regex.hpp>
#include <string>
#include <iostream>

using namespace boost::algorithm;

int main()
{
  std::string s = "Boost C++ Libraries";
  boost::iterator_range<std::string::iterator> r =
    find_regex(s, boost::regex{"\\w\\+\\+"});
  std::cout << r << '\n';
}
```

In order to use the regular expression, the program accesses a class called `boost::regex`, which is presented in Chapter 8.

Example 5.12 writes `C++` to standard output.

Exercise

Create a program which asks the user to enter his full name. The program should greet the user with “Hello” followed by the user's name followed by an exclamation mark. The user's first- and lastname should start with a capital letter followed by lowercase letters. Furthermore, the user's first- und lastname should be separated with exactly one space. There should be no space before the exclamation mark.

Solutions