Chapter 10. Boost.Tokenizer

The library Boost.Tokenizer allows you to iterate over partial expressions in a string by interpreting certain characters as separators.

Example 10.1. Iterating over partial expressions in a string with `boost::tokenizer`

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

int main()
{
  typedef boost::tokenizer<boost::char_separator<char>> tokenizer;
  std::string s = "Boost C++ Libraries";
  tokenizer tok{s};
  for (tokenizer::iterator it = tok.begin(); it != tok.end(); ++it)
    std::cout << *it << '\n';
}
```

Boost.Tokenizer defines a class template called `boost::tokenizer` in `boost/tokenizer.hpp`. It expects as a template parameter a class that identifies coherent expressions. Example 10.1 uses the class `boost::char_separator`, which interprets spaces and punctuation marks as separators.

A tokenizer must be initialized with a string of type `std::string`. Using the member functions `begin()` and `end()`, the tokenizer can be accessed like a container. Partial expressions of the string used to initialize the tokenizer are available via iterators. How partial expressions are evaluated depends on the kind of class passed as the template parameter.

Because `boost::char_separator` interprets spaces and punctuation marks as separators by default, Example 10.1 displays `Boost`, `C`, `+`, `+`, and `Libraries`. `boost::char_separator` uses `std::isspace()` and `std::ispunct()` to identify separator characters. Boost.Tokenizer distinguishes between separators that should be displayed and separators that should be suppressed. By default, spaces are suppressed and punctuation marks are displayed.

Example 10.2. Initializing `boost::char_separator` to adapt the iteration

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

int main()
{
  typedef boost::tokenizer<boost::char_separator<char>> tokenizer;
  std::string s = "Boost C++ Libraries";
  boost::char_separator<char> sep{" "};
  tokenizer tok{s, sep};
  for (const auto &t : tok)
    std::cout << t << '\n';
}
```

To keep punctuation marks from being interpreted as separators, initialize the `boost::char_separator` object before passing it to the tokenizer.

The constructor of `boost::char_separator` accepts a total of three parameters, but only the first one is required. The first parameter describes the individual separators that are suppressed. Example 10.2, like Example 10.1, treats spaces as separators.

The second parameter specifies the separators that should be displayed. If this parameter is omitted, no separators are displayed, and the program will now display `Boost`, `C++` and `Libraries`.

Example 10.3. Simulating the default behavior with `boost::char_separator`

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

int main()
{
  typedef boost::tokenizer<boost::char_separator<char>> tokenizer;
  std::string s = "Boost C++ Libraries";
  boost::char_separator<char> sep{" ", "+"};
  tokenizer tok{s, sep};
  for (const auto &t : tok)
    std::cout << t << '\n';
}
```

If a plus sign is passed as the second parameter, Example 10.3 behaves like Example 10.1.

The third parameter determines whether or not empty partial expressions are displayed. If two separators are found back-to-back, the corresponding partial expression is empty. By default, these empty expressions are not displayed. Using the third parameter, the default behavior can be changed.

Example 10.4. Initializing `boost::char_separator` to display empty partial expressions

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

int main()
{
  typedef boost::tokenizer<boost::char_separator<char>> tokenizer;
  std::string s = "Boost C++ Libraries";
  boost::char_separator<char> sep{" ", "+", boost::keep_empty_tokens};
  tokenizer tok{s, sep};
  for (const auto &t : tok)
    std::cout << t << '\n';
}
```

Example 10.4 displays two additional empty partial expressions. The first one is found between the two plus signs, while the second one is found between the second plus sign and the following space.

Example 10.5. Boost.Tokenizer with wide strings

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

int main()
```

```
{
  typedef boost::tokenizer<boost::char_separator<wchar_t>,
    std::wstring::const_iterator, std::wstring> tokenizer;
  std::wstring s = L"Boost C++ Libraries";
  boost::char_separator<wchar_t> sep{L" "};
  tokenizer tok{s, sep};
  for (const auto &t : tok)
    std::wcout << t << '\n';
}
```

Example 10.5 iterates over a string of type `std::wstring`. In order to support this string type, the tokenizer must be initialized with additional template parameters. The class `boost::char_separator` must also be initialized with `wchar_t`.

Besides `boost::char_separator`, Boost.Tokenizer provides two additional classes to identify partial expressions.

Example 10.6. Parsing CSV files with `boost::escaped_list_separator`

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

int main()
{
  typedef boost::tokenizer<boost::escaped_list_separator<char>> tokenizer;
  std::string s = "Boost,\"C++ Libraries\"";
  tokenizer tok{s};
  for (const auto &t : tok)
    std::cout << t << '\n';
}
```

`boost::escaped_list_separator` is used to read multiple values separated by commas. This format is commonly known as CSV (Comma Separated Values). `boost::escaped_list_separator` also handles double quotes and escape sequences. Therefore, the output of Example 10.6 is `Boost` and `C++ Libraries`.

The second class provided is `boost::offset_separator`, which must be instantiated. The corresponding object must be passed to the constructor of `boost::tokenizer` as a second parameter.

Example 10.7. Iterating over partial expressions with `boost::offset_separator`

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

int main()
{
  typedef boost::tokenizer<boost::offset_separator> tokenizer;
  std::string s = "Boost_C++_Libraries";
  int offsets[] = {5, 5, 9};
  boost::offset_separator sep{offsets, offsets + 3};
  tokenizer tok{s, sep};
  for (const auto &t : tok)
    std::cout << t << '\n';
}
```

`boost::offset_separator` specifies the locations within the string where individual partial expressions end. Example 10.7 specifies that the first partial expression ends after 5 characters, the second ends after an additional 5 characters, and the third ends after the following 9 characters. The output will be `Boost`, `_C++_` and `Libraries`.