

## 23.4 — Stream classes for strings

ALEX DECEMBER 21, 2020

So far, all of the I/O examples you have seen have been writing to `cout` or reading from `cin`. However, there is another set of classes called the stream classes for strings that allow you to use the familiar insertions (`<<`) and extraction (`>>`) operators to work with strings. Like `istream` and `ostream`, the string streams provide a buffer to hold data. However, unlike `cin` and `cout`, these streams are not connected to an I/O channel (such as a keyboard, monitor, etc...). One of the primary uses of string streams is to buffer output for display at a later time, or to process input line-by-line.

There are six stream classes for strings: `istringstream` (derived from `istream`), `ostringstream` (derived from `ostream`), and `stringstream` (derived from `iostream`) are used for reading and writing normal characters width strings. `wstringstream`, `wostringstream`, and `wstringstream` are used for reading and writing wide character strings. To use the stringstreams, you need to `#include` the `sstream` header.

There are two ways to get data into a `stringstream`:

1. Use the insertion (`<<`) operator:

```
1 | std::stringstream os;
  | os << "en garde!" << '\n'; // insert "en garde!" into the
  | stringstream
```

2. Use the `str(string)` function to set the value of the buffer:

```
1 | std::stringstream os;
  | os.str("en garde!"); // set the stringstream buffer to "en
  | garde!"
```

There are similarly two ways to get data out of a `stringstream`:

1. Use the `str()` function to retrieve the results of the buffer:

```
1 | std::stringstream os;
  | os << "12345 67.89" <<
  | '\n';
  | cout << os.str();
```

This prints:

```
12345 67.89
```

2. Use the extraction (`>>`) operator:

```
1 | std::stringstream os;
  | os << "12345 67.89"; // insert a string of numbers into the
  | stream
  |
  | std::string strValue;
  | os >> strValue;
  |
  | std::string strValue2;
  | os >> strValue2;
  |
  | // print the numbers separated by a dash
  | std::cout << strValue << " - " << strValue2 << '\n';
```

This program prints:

```
12345 - 67.89
```

Note that the >> operator iterates through the string -- each successive use of >> returns the next extractable value in the stream. On the other hand, str() returns the whole value of the stream, even if the >> has already been used on the stream.

### Conversion between strings and numbers

Because the insertion and extraction operators know how to work with all of the basic data types, we can use them in order to convert strings to numbers or vice versa.

First, let's take a look at converting numbers into a string:

```
1 std::stringstream os;
2 int nValue{ 12345 };
3 double dValue{ 67.89 };
  os << nValue << ' ' << dValue;
4
5 std::string strValue1, strValue2;
  os >> strValue1 >> strValue2;
6
7 std::cout << strValue1 << ' ' << strValue2 <<
  '\n';
```

This snippet prints:

```
12345 67.89
```

Now let's convert a numerical string to a number:

```
1 std::stringstream os;
  os << "12345 67.89"; // insert a string of numbers into the
2 stream
  int nValue;
  double dValue;
3
  os >> nValue >> dValue;
4
  cout << nValue << ' ' << dValue << '\n';
```

This program prints:

```
12345 67.89
```

Clearing a stringstream for reuse

There are several ways to empty a stringstream's buffer.

1. Set it to the empty string using `str()` with a blank C-style string:

```
1 | std::stringstream os;  
  | os << "Hello ";  
2 |  
3 | os.str(""); // erase the  
4 | buffer  
5 |  
6 | os << "World!";  
7 | std::cout << os.str();
```

2. Set it to the empty string using `str()` with a blank `std::string` object:

```
1 | std::stringstream os;  
2 | os << "Hello ";  
3 |  
4 | os.str(std::string{}); // erase the  
  | buffer  
5 |  
6 | os << "World!";  
7 | cout << os.str();
```

Both of these programs produce the following result:

World!

When clearing out a stringstream, it is also generally a good idea to call the `clear()` function:

```
1  std::stringstream os;  
   os << "Hello ";  
2  
3  os.str(""); // erase the  
4  buffer  
   os.clear(); // reset error  
5  flags  
  
6  os << "World!";  
7  std::cout << os.str();  
8
```

`clear()` resets any error flags that may have been set and returns the stream back to the ok state. We will talk more about the stream state and error flags in the next lesson.



## Next lesson

**23.5** Stream states and input validation



[Back to table of contents](#)



## Previous lesson

**23.3** Output with ostream and ios

Leave a comment... Put C++ code between triple-backticks (markdown style): ``Your C++ code``

 Name\*

@ Email\* ?

Avatars from <https://gravatar.com/> are connected to your provided email address.

Notify me about replies:



POST COMMENT

35 COMMENTS

Newest ▼

