28.4 — Stream classes for strings

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. wistringstream, 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:

```
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:

```
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:

```
std::stringstream os {};
os << "12345 67.89\n";
std::cout << os.str();
```

This prints:

```
12345 67.89
```

2. Use the extraction (>>) operator:

```
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:

```
std::stringstream os {};

constexpr int nValue { 12345 };
constexpr double dValue { 67.89 };
os << nValue << ' ' << dValue;

std::string strValue1, strValue2;
os >> strValue1 >> strValue2;

std::cout << strValue1 << ' ' << strValue2 << '\n';
```

This snippet prints:

```
12345 67.89
```

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

```
std::stringstream os {};
os << "12345 67.89"; // insert a string of numbers into the stream
int nValue {};
double dValue {};

os >> nValue >> dValue;

std::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:

```
std::stringstream os {};
os << "Hello ";

os.str(""); // erase the buffer

os << "World!";
std::cout << os.str();
```

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

```
std::stringstream os {};
os << "Hello ";

os.str(std::string{}); // erase the buffer

os << "World!";
std::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:

```
std::stringstream os {};
os << "Hello ";

os.str(""); // erase the buffer
os.clear(); // reset error flags

os << "World!";
std::cout << os.str();
```

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.