28.4 — Stream classes for strings

ALEX[1] FEBRUARY 27, 2024

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.

Next lesson
28.5 Stream states and input validation

2

Back to table of contents

3

Previous lesson
28.3 Output with ostream and ios

4

5

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[6] are connected to your provided email address.

58 COMMENTS

Newest

Nidhi Gupta
February 23, 2025 9:48 pm PST

C++ provides special string stream classes—namely istringstream, ostringstream, and stringstream (and their wide-character versions)—that allow you to use the usual insertion (<<) and extraction (>>) operators on strings. Unlike ordinary I/O streams like cin and cout that are linked to external devices, these string streams possess an internal buffer to store the data. You may insert data into a string stream either using the insertion operator or by directly initializing the buffer with str(). Similarly, you may retrieve data by reading sequential tokens using the extraction operator or by reading the entire buffer with str(). This is especially useful for conversion from string-to-other-data-type and for input processing or buffering output to be used in the future. Also, string streams can be cleared and reused

by releasing their buffer as an empty string, making them a useful tool for any given programming environment.

0 Reply

Leni
February 23, 2025 6:05 pm PST

String streams provide a flexible way to handle string-based input and output operations using familiar stream operators. How do the insertion (<<) and extraction (>>) operators differ when working with string streams compared to standard I/O streams like `cin` and `cout`?

1 Reply

Timmy
Reply to Leni [7] March 18, 2025 9:55 am PDT

The moment when you lowk have the same question but you see some1 else asked your question but it was left unanswered and its not swag. You feel me? Sorry for notifying your gmail and its some dumb*typing

0 Reply

EmtyC
January 4, 2025 7:31 am PST

Why doesn't the STL provide a std::iostream object ? Is the only purpose of std::iostream being a base class to std::stringstream and std::fstream ?

1 Reply

Alex Author
Reply to EmtyC [8] January 17, 2025 8:23 pm PST

Because std::cin, std::cout, and std::cerr are designed to parallel the standard streams. See https://en.wikipedia.org/wiki/Standard_streams

I'm not sure what std::iostream would represent. Something that can do both input and output? If it's possible, then perhaps there wasn't a need...

2 Reply

EmtyC
Reply to Alex [9] January 18, 2025 7:20 am PST

Oh cool. Thanks :D

0 Reply

Jestin PJ

September 11, 2023 3:22 am PDT

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

int main() {
    std::vector<std::string> extractedStrings;
    std::string str;
    std::stringstream os;

    std::cout << "Enter some text: ";
    std::getline(std::cin, str);
    os << str;
    str.erase();

    while (os >> str) {
        extractedStrings.push_back(str);
    }

    // Print the extracted strings
    for (const std::string& extractedStr : extractedStrings) {
        std::cout << "Extracted: " << extractedStr << std::endl;
    }

    return 0;
}
```

Last edited 1 year ago by Jestin PJ

0 Reply

learnccp lesson reviewer

August 2, 2023 7:00 pm PDT

Awesome lesson! Knowledge aquired.

1 Reply

Emeka Daniel

May 27, 2023 5:46 am PDT

I get it now,

objects cout and cin, use two distinct streams, cout uses the output stream while cin uses the input stream.

When cout is used in a program like this:

```cpp
Using namespace std;

cout << "Daniel is Awesome" << '\n';
```

cout inserts the following text into the output stream in a buffered like manner waiting for which ever output device to display said text(in this case the monitor).

cin is a little bit different.
When used as so:

```
1 Using namespace std;
2
3 int a;
4
5 cin >> a;
```

cin extracts input stored in a buffer like manner in the input stream to whatever object given. Like so, when a user enters certain keys to the console, they are all inserted in the input stream[buffered] awaiting extraction by cin.

This also explains why input fails.
Cuz as you told us, a stream size can be sometimes infinite, that means it has no definitive size[I think, correct if wrong], so cin doesn't have prior knowledge of the type of object it is extracting to, so when for instance in my example above I insert a character 'd', it gets stored in the input stream, where cin then extracts said character to object of type int, it causes a failure in extraction, because an object of type int won't have a character value.

Now, stringstream is based on the same concept as cin and cout, but instead of separate streams they use they same stream and instead of close coordination with various input and output devices, they coordinate instead with objects of type string.

They also Inherit specific traits, like how extraction from the input stream breaks at Whitespace, extraction from the string stream also breaks at Whitespace.

This is how I understand it, correct any area I misconstrued. Thank you.

0 Reply

Emeka Daniel

Reply to Emeka Daniel [10] May 28, 2023 7:35 am PDT

The above explanation is deeply flawed, pls pay no attention to it.

0 Reply

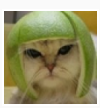
Emeka Daniel

Reply to Emeka Daniel [10] May 27, 2023 6:46 am PDT

Question:

1. Will it be correct to define a stream as a stream of bytes that isn't defined by a specific data type.

0 Reply



Alex Author

Reply to Emeka Daniel [11] May 30, 2023 5:25 pm PDT

No, a stream could be a stream of elements of some data type.

0 Reply

Emeka Daniel

Reply to Alex [12] May 31, 2023 2:07 am PDT

Okay, but don't streams used by cin and cout have no specific data type?

0 Reply

Alex Author

Reply to Emeka Daniel [13] June 2, 2023 3:38 pm PDT

No, they have to have a specific data type. I think std::cin uses `char`, so it reads a stream of chars.

When you extract values, those sequences of chars can be converted to other types.

0 Reply

Emeka Daniel

Reply to Alex [14] June 2, 2023 4:22 pm PDT

Ohh, okay. Nice to know

0 Reply

Waldo

July 23, 2022 7:56 am PDT

> `" << '\n'`

* `\n"`

0 Reply

Blake

June 1, 2022 2:34 pm PDT

The example codes does not work without `#includ`ing `<sstream>` for me. I had to google the error to fix it.(Edit- Sorry I didn't read carefully.

4 Reply

mmpizza

February 14, 2022 8:16 am PST

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

should be std::cout

1 Reply

tony

February 3, 2022 10:34 pm PST

Could I use `"en garde!" >> os` to insert value to `os` ?

1 Reply

Alex Author

Reply to tony [15] February 4, 2022 10:53 am PST

No, `operator>>` is used to extract values from the stream, not insert them.

2 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/stream-states-and-input-validation/
3. https://www.learncpp.com/
4. https://www.learncpp.com/cpp-tutorial/output-with-ostream-and-ios/
5. https://www.learncpp.com/stream-classes-for-strings/
6. https://gravatar.com/
7. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-608001
8. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-606219
9. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-606690
10. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-580859
11. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-580860
12. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-581005
13. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-581028
14. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-581104
15. https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/#comment-565061
16. https://g.ezoic.net/privacy/learncpp.com