22.2 — std::string construction and destruction

ALEX JUNE 15, 2023

In this lesson, we'll take a look at how to construct objects of std::string, as well as how to create strings from numbers and vice-versa.

String construction

The string classes have a number of constructors that can be used to create strings. We'll take a look at each of them here.

Note: string::size_type resolves to size_t, which is the same unsigned integral type that is returned by the sizeof operator. The actual size of size_t depending on the environment. For the purposes of this tutorial, envision it as an unsigned int.

string::string()

- This is the default constructor. It creates an empty string.

Sample code:

```
std::string sSource;
std::cout << sSource;
```

Output:

```

```

string::string(const string& strString)

- This is the copy constructor. This constructor creates a new string as a copy of strString.

Sample code:

```
std::string sSource{ "my string" };
std::string sOutput{ sSource };
std::cout << sOutput;
```

Output:

```
my string
```

string::string(const string& strString, size_type unIndex)
string::string(const string& strString, size_type unIndex, size_type unLength)

- This constructor creates a new string that contains at most unLength characters from strString, starting with index unIndex. If a NULL is encountered, the string copy will end, even if unLength has not been reached.
- If no unLength is supplied, all characters starting from unIndex will be used.
- If unIndex is larger than the size of the string, the out_of_range exception will be thrown.

Sample code:

```
    std::string sSource{ "my string" };
    std::string sOutput{ sSource, 3 };
    std::cout << sOutput<< '\n';
    std::string sOutput2(sSource, 3, 4);
    std::cout << sOutput2 << '\n';
```

Output:

```
string
stri
```

string::string(const char* szCString)

- This constructor creates a new string from the C-style string szCString, up to but not including the NULL terminator.
- If the resulting size exceeds the maximum string length, the length_error exception will be thrown.
- Warning: szCString must not be NULL.

Sample code:

```
    const char* szSource{ "my string" };
    std::string sOutput{ szSource };
    std::cout << sOutput << '\n';
```

Output:

```
my string
```

string::string(const char* szCString, size_type unLength)

- This constructor creates a new string from the first unLength chars from the C-style string szCString.
- If the resulting size exceeds the maximum string length, the length_error exception will be thrown.
- Warning: For this function only, NULLs are not treated as end-of-string characters in szCString! This means it is possible to read off the end of your string if unLength is too big. Be careful not to overflow your string buffer!

Sample code:

```
    const char* szSource{ "my string" };
    std::string sOutput(szSource, 4);
    std::cout << sOutput << '\n';
```

Output:

```
my s
```

string::string(size_type nNum, char chChar)

- This constructor creates a new string initialized by nNum occurances of the character chChar.
- If the resulting size exceeds the maximum string length, the length_error exception will be thrown.

Sample code:

```
    std::string sOutput(4, 'Q');
    std::cout << sOutput << '\n';
```

Output:

```
QQQQ
```

template string::string(InputIterator itBeg, InputIterator itEnd)

- This constructor creates a new string initialized by the characters of range [itBeg, itEnd).
- If the resulting size exceeds the maximum string length, the length_error exception will be thrown.

No sample code for this one. It's obscure enough you'll probably never use it.

string::~string()

String destruction

- This is the destructor. It destroys the string and frees the memory.

No sample code here either since the destructor isn't called explicitly.

Constructing strings from numbers

One notable omission in the std::string class is the lack of ability to create strings from numbers. For example:

```
        std::string sFour{ 4 };
```

Produces the following error:

```
c:vcprojectstest2test2test.cpp(10) : error C2664: 'std::basic_string<_Elem,_Traits,_Ax>::basic_string(std::
basic_string<_Elem,_Traits,_Ax>::_Has_debug_it)' : cannot convert parameter 1 from 'int' to 'std::basic_str
ing<_Elem,_Traits,_Ax>::_Has_debug_it'
```

Remember what I said about the string classes producing horrible looking errors? The relevant bit of information here is:

```
cannot convert parameter 1 from 'int' to 'std::basic_string
```

In other words, it tried to convert your int into a string but failed.

The easiest way to convert numbers into strings is to involve the std::ostringstream class. std::ostringstream is already set up to accept input from a variety of sources, including characters, numbers, strings, etc... It is also capable of outputting strings (either via the extraction operator>>, or via the str() function). For more information on std::ostringstream, see 28.4 -- Stream classes for strings (https://www.learncpp.com/cpp-tutorial/stream-classes-for-strings/).

Here's a simple solution for creating std::string from various types of inputs:

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

template <typename T>
inline std::string ToString(T tX)
{
    std::ostringstream oStream;
    oStream << tX;
    return oStream.str();
}
```

Here's some sample code to test it:

```
int main()
{
    std::string sFour{ ToString(4) };
    std::string sSixPointSeven{ ToString(6.7) };
    std::string sA{ ToString('A') };
    std::cout << sFour << '\n';
    std::cout << sSixPointSeven << '\n';
    std::cout << sA << '\n';
}
```

And the output:

```
4
6.7
A
```

Note that this solution omits any error checking. It is possible that inserting tX into oStream could fail. An appropriate response would be to throw an exception if the conversion fails.

Related content

The standard library also contains a function named `std::to_string()` that can be used to convert numbers into a std::string. While this is a simpler solution for basic cases, the output of std::to_string may differ from the output of std::cout or our ToString() function above. Some of these differences are currently documented here (https://en.cppreference.com/w/cpp/string/basic_string/to_string).

Converting strings to numbers

Similar to the solution above:

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

template <typename T>
inline bool FromString(const std::string& sString, T& tX)
{
    std::istringstream iStream(sString);
    return !(iStream >> tX).fail(); // extract value into tX, return success or not
}
```

Here’s some sample code to test it:

```
int main()
{
    double dX;
    if (FromString("3.4", dX))
        std::cout << dX << '\n';
    if (FromString("ABC", dX))
        std::cout << dX << '\n';
}
```

And the output:

```
3.4
```

Note that the second conversion failed and returned false.

Next lesson
22.3 std::string length and capacity

Back to table of contents

Previous lesson
22.1 std::string and std::wstring

Leave a comment...