Converting a C-style string to a C++ std::string

What is the best way to convert a C-style string to a C++ `std::string` ? In the past I've done it using `stringstream` s. Is there a better way?

c++ string cstring

edited Jul 11 '13 at 1:27 templatetypedef **215k** 51 533 793

asked Jan 21 '11 at 23:23 blcArmadillo **2,474** 12 40 75

What's a cstring? Do you mean a `CString` from MFC? Or a null-terminated array of char (a C string)? Or something else? – Rob Kennedy Jan 21 '11 at 23:43

6 Answers

C++ strings have a constructor that lets you convert C-style strings:

```
char* myStr = "This is a C string!";
std::string myCppString = myStr;
```

answered Jan 21 '11 at 23:25 templatetypedef **215k** 51 533 793

and now I also have to do `delete myStr;` no? – Barnabas Szabolcs Nov 6 '15 at 14:23

@BarnabasSzabolcs No, that's not necessary. You only need to delete memory allocated with new. Pointers to string literals don't need to be deallocated. – templatetypedef Nov 6 '15 at 16:07

1 Every answer here fails to mention the obvious edge case. If your char* is NULL, std::string will throw. It will not be an empty string as many would suspect. It's unfortunate that all the top posts on stackoverflow don't mention this, and I suspect many people who google for this simple conversion are dealing with the bugs later. – Trevor Hickey Nov 11 '15 at 13:02

1 @TrevorHickey While that's true, one could argue that NULL isn't a string. It's the absence of a string. – templatetypedef Nov 11 '15 at 16:30

1 @templatetypedef Agreed. The answers here aren't wrong, but a disclaimer about NULL would go a long way in terms of helping others. There are many common functions("getenv()" for example), that may or may not return NULL when called with the same inputs. By giving newcomers a simple one-liner without adding a disclaimer is setting them up for failure. – Trevor Hickey Nov 11 '15 at 17:02

Check the different constructors of the string class: documentation You maybe interested in:

```
//string(char* s)
std::string str(cstring);
```

And:

```
//string(char* s, size_t n)
std::string str(cstring, len_str);
```

edited Aug 28 '13 at 17:26
user283145

answered Jan 21 '11 at 23:28

Santiago Alessandri
4,081 17 36

You can initialise a `std::string` directly from a c-string:

```
std::string s = "i am a c string";
std::string t = std::string("i am one too");
```

answered Jan 21 '11 at 23:25

trojanfoe
94.3k 12 143 180

If you mean `char*` to `std::string`, you can use the constructor.

```
char* a;
std::string s(a);
```

Or if the `string s` already exist, simply write this:

```
s=std::string(a);
```

answered Jul 11 '13 at 1:40

Manas
497 2 12

No. Your example would throw a logic error in std::string's constructor. 'a' cannot be NULL. – Trevor Hickey Nov 11 '15 at 13:04

In general (without declaring new storage) you can just use the 1-arg constructor to change the c-string into a string rvalue :

```
string xyz = std::string("this is a test") +
             std::string(" for the next 60 seconds ") +
             std::string("of the emergency broadcast system.");
```

However, this does not work when constructing the string to pass it by reference to a function (a problem I just ran into), e.g.

```
void ProcessString(std::string& username);
ProcessString(std::string("this is a test"));   // fails
```

You need to make the reference a const reference:

```
void ProcessString(const std::string& username);
ProcessString(std::string("this is a test"));   // works.
```

answered May 9 '14 at 20:51

user216843
11 2

C++11 : Overload a string literal operator

```
std::string operator ""_s(const char * str, std::size_t len) {
    return std::string(str, len);
}

auto s1 = "abc\0\0def";     // C style string
auto s2 = "abc\0\0def"_s;   // C++ style std::string
```

C++14 : Use the operator from `std::string_literals` namespace

```
using namespace std::string_literals;

auto s3 = "abc\0\0def"s;    // is a std::string
```

answered Jul 8 '15 at 12:24

Shreevardhan
4,314 2 13 35
