Prepend std::string

What is the most efficient way to prepend `std::string`? Is it worth writing out an entire function to do so, or would it take only 1 - 2 lines? I'm not seeing anything related to an `std::string::push_front`.

c++ string prepend

asked Dec 12 '11 at 0:15
zeboidlund
2,819 ● 10 ● 56 ● 122

1 Any reason you can't do something like `s = a + s`? – Mike Bantegui Dec 12 '11 at 0:16

What do u mean by prepend? Is it what @MikeBantegui has mentioned? or you are trying to do something else? – Ankit Dec 12 '11 at 0:18

Prepending to a string is not going to be efficient, perhaps it may be better to append to the string and reverse it when you are finished? – GWW Dec 12 '11 at 0:19

@GWW: Is that really more efficient? – Lightness Races in Orbit Dec 12 '11 at 0:21

Do you mean prepend *to* a `std::string`? Then prepend what? – Lightness Races in Orbit Dec 12 '11 at 0:21

4 Answers

There actually is a similar function to the non-existing `std::string::push_front`, see the below example.

Documentation of std::string::insert

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

int
main (int argc, char *argv[])
{
  std::string s1 (" world");
  std::string s2 ("ello");

  s1.insert (0,     s2); // insert the contents of s2 at offset 0 in s1
  s1.insert (0, 1, 'h'); // insert one (1) 'h'        at offset 0 in s1

  std::cout << s1 << std::endl;
}
```

output:

```
hello world
```

Since prepending a string with data might require both reallocation and copy/move of existing data you can get some performance benefits by getting rid of the reallocation part by using `std::string::reserve` (to allocate more memory before hand).

The copy/move of data is sadly quite inevitable, unless you define your own custom made class that acts like `std::string` that allocates a large buffer and places the first content in the center of this memory buffer.

Then you can both prepend and append data without reallocation and moving data, if the buffer is large enough that is. Copying from *source* to *destination* is still, obviously, required though.

If you have a buffer in which you know you will *prepend* data more often than you *append* a good alternative is to store the string backwards, and reversing it when needed (if that is more rare).

edited Aug 28 '13 at 19:36
user283145

answered Dec 12 '11 at 0:24
Filip Roséen - refp
30.8k ● 9 ● 59 ● 121

I also want to add that if you need to prepend a string with multiple items, you can avoid quadratic-time performance by first appending the items and then rotating them to the front using `std::rotate` from <algorithm>. – Don Reba Dec 12 '11 at 1:06

There is an overloaded `string operator+ (char lhs, const string& rhs);` , so you can just do `your_string 'a' + your_string` to mimic `push_front` .

This is not in-place but creates a new string, so don't expect it to be efficient, though. For a (probably) more efficient solution, use `resize` to gather space, `std::copy_backward` to shift the entire string back by one and insert the new character at the beginning.

answered Dec 12 '11 at 0:18
Alexander Gessler
30.6k ● 2 ● 45 ● 91

You should **not** use `std::string::resize` to "gather more space", the function will make the internal buffer bigger if required, that's true. But it will also pad the current string with `char()` 's (also known as `NULL` (ie. the value 0). See this codepad paste for a demonstration. – Filip Roséen - refp Dec 12 '11 at 0:33

If you're using `std::string::append` , you should realize the following is equivalent:

```
std::string lhs1 = "hello ";
std::string lh2 = "hello ";
std::string rhs = "world!";

lhs1.append(rhs);
lhs2 += rhs; // equivalent to above
// Also the same:
// lhs2 = lhs + rhs;
```

Similarly, a "prepend" would be equivalent to the following:

```
std::string result = "world";
result = "hello " + result;
// If prepend existed, this would be equivalent to
// result.prepend("hello");
```

You should note that it's rather inefficient to do the above though.

answered Dec 12 '11 at 0:18
Mike Bantegui
6,335 ● 4 ● 34 ● 90

```
myString.insert(0, otherString);
```

Let the Standard Template Library writers worry about efficiency; make use of all their hours of work rather than re-programming the wheel.

This way does both of those.

As long as the STL implementation you are using was thought through you'll have efficient code. If you're using a badly written STL, you have bigger problems anyway :)

answered Jan 27 '14 at 11:11
matiu
2,769 ● 16 ● 28

An inefficient algorithm can't be rescued by a smart implementation. – Ben Voigt Sep 25 '14 at 13:35