C++ string manipulation

Asked 16 years, 1 month ago Modified 13 years, 5 months ago Viewed 4k times

6

My lack of C++ experience, or rather my early learning in garbage collected languages is really stinging me at the moment and I have a problem working with strings in C++.

To make it very clear, using std::string or equlivents is not an option - this is char* 's all the way.

So: what I need to do is very simple and basically boils down to concatenating strings. At runtime I have 2 classes.

One class contains "type" information in the form of a base filename.

in the header:

```
char* mBaseName;
```

and later, in the .cpp it is loaded with info passed in from elsewhere.

```
mBaseName = attributes->BaseName;
```

The 2nd class provides version information in the form of a suffix to the base file name, it's a static class and implemented like this at present:

```
static const char* const suffixes[] = {"Version1", "Version", "Version3"};
//etc.

static char* GetSuffix()
{
    int i = 0;
    //perform checks on some data structures
    i = somevalue;
   return suffixes[i];
}
```

Then, at runtime the base class creates the filename it needs:

```
void LoadStuff()
{
    char* suffix = GetSuffix();
    char* nameToUse = new char[50];
    sprintf(nameToUse, "%s%s",mBaseName,suffix);
```

```
    LoadAndSetupData(nameToUse);
}
```

And you can see the problem immediately. nameToUse never gets deleted, memory leak.

The suffixes are a fixed list, but the basefilenames are arbitrary. The name that is created needs to persist beyond the end of "LoadStuff()" as it's not clear when if and how it is used subsequently.

I am probably worrying too much, or being very stupid, but similar code to LoadStuff() happens in other places too, so it needs solving. It's frustrating as I don't quite know enough about the way things work to see a safe and "un-hacky" solution. In C# I'd just write:

```
LoadAndSetupData(mBaseName + GetSuffix());
```

and wouldn't need to worry.

Any comments, suggestions, or advice much appreciated.

Update

The issue with the code I am calling LoadAndSetupData() is that, at some point it probably does copy the filename and keep it locally, but the actual instantiation is asynchranous, LoadAndSetupData actually puts things into a queue, and at that point at least, it expects that the string passed in still exists.

I do not control this code so I can't update it's function.

c++ c string character

Share
Improve this question
Follow

edited Jul 18, 2011 at 9:32

asked Nov 10, 2008 at 16:31

xan
7,568 ● 8 ● 46 ● 70

Do you have some way of detecting when the string has been pulled off the queue? – Eclipse Nov 10, 2008 at 16:52

You should re-tag this question "C string manipulation" if you're using C style strings and can't use common C++ language features such as std::string. – Aaron Dec 4, 2008 at 1:15

13 Answers

Sorted by: Highest score (default)

3

Seeing now that the issue is how to clean up the string that you created and passed to LoadAndSetUpData()

I am assuming that:

1. LoadAndSetUpData() does not make its own copy
2. You can't change LoadAndSetUpData() to do that
3. You need the string to still exist for some time after LoadAndSetupData() returns

Here are suggestions:

1. Can you make your own queue objects to be called? Are they guaranteed to be called after the ones that use your string. If so, create cleanup queue events with the same string that call delete[] on them
2. Is there a maximum number you can count on. If you created a large array of strings, could you use them in a cycle and be assured that when you got back to the beginning, it would be ok to reuse that string
3. Is there an amount of time you can count on? If so, register them for deletion somewhere and check that after some time.

The best thing would be for functions that take char* to take ownership or copy. Shared ownership is the hardest thing to do without reference counting or garbage collection.

Share Improve this answer Follow

answered Nov 10, 2008 at 16:56

Lou Franco

89k ● 14 ● 136 ● 198

3

EDIT: This answer doesn't address his problem completely -- I made other suggestions here: C++ string manipulation

His problem is that he needs to extend the scope of the char* he created to outside the function, and until an asynchronous job is finished.

Original Answer:

In C++, if I can't use the standard library or Boost, I still have a class like this:

```
template<class T>
class ArrayGuard {
  public:
```

```
    ArrayGuard(T* ptr) { _ptr = ptr; }
    ~ArrayGuard() { delete[] _ptr; }
  private:
    T* _ptr;
    ArrayGuard(const ArrayGuard&);
    ArrayGuard& operator=(const ArrayGuard&);
}
```

You use it like:

```
char* buffer = new char[50];
ArrayGuard<char *> bufferGuard(buffer);
```

The buffer will be deleted at the end of the scope (on return or throw).

For just simple array deleting for dynamic sized arrays that I want to be treated like a static sized array that gets released at the end of the scope.

Keep it simple -- if you need fancier smart pointers, use Boost.

This is useful if the 50 in your example is variable.

Share

Improve this answer

Follow

edited May 23, 2017 at 12:08

Community Bot

1 • 1

answered Nov 10, 2008 at 16:44

Lou Franco

89k • 14 • 136 • 198

Or if your platform doesn't support templates, just make a char-specific class. – Eclipse Nov 10, 2008 at 16:49

Your solution has the drawback that it cannot be copied. so if it goes out of scope, the memory will be freed, and the copy handed over to LoadAndSetupData is a dangling reference - just seen you assume it doesn't copy. Indeed then it works – Johannes Schaub - litb Nov 10, 2008 at 17:09

@Lou, Your solution is a poor approzimation of std::basic_string. If you need fancier strings, use string. *sigh* – Aaron Dec 4, 2008 at 1:18

3

The thing to remember with C++ memory management is ownership. If the LoadAndSetupData data is not going to take ownership of the string, then it's still your responsibility. Since you can't delete it immediately (because of the asynchronicity issue), you're going to have to hold on to those pointers until such time as you know you can delete them.

Maintain a pool of strings that you have created:

- If you have some point in time where you know that the queue has been completely dealt with, you can simply delete all the strings in the pool.

- If you know that all strings created after a certain point in time have been dealt with, then keep track of when the strings were created, and you can delete that subset. - If you can somehow find out when an individual string has been dealt with, then just delete that string.

```
class StringPool
{
    struct StringReference {
        char *buffer;
        time_t created;
    } *Pool;

    size_t PoolSize;
    size_t Allocated;

    static const size_t INITIAL_SIZE = 100;

    void GrowBuffer()
    {
        StringReference *newPool = new StringReference[PoolSize * 2];
        for (size_t i = 0; i < Allocated; ++i)
            newPool[i] = Pool[i];
        StringReference *oldPool = Pool;
        Pool = newPool;
        delete[] oldPool;
    }

public:

    StringPool() : Pool(new StringReference[INITIAL_SIZE]),
PoolSize(INITIAL_SIZE)
    {
    }

    ~StringPool()
    {
        ClearPool();
        delete[] Pool;
    }

    char *GetBuffer(size_t size)
    {
        if (Allocated == PoolSize)
            GrowBuffer();
        Pool[Allocated].buffer = new char[size];
        Pool[Allocated].buffer = time(NULL);
        ++Allocated;
    }

    void ClearPool()
    {
        for (size_t i = 0; i < Allocated; ++i)
            delete[] Pool[i].buffer;
        Allocated = 0;
    }

    void ClearBefore(time_t knownCleared)
    {
```

```
        size_t newAllocated = 0;
        for (size_t i = 0; i < Allocated; ++i)
        {
            if (Pool[i].created < knownCleared)
            {
                delete[] Pool[i].buffer;
            }
            else
            {
                Pool[newAllocated] = Pool[i];
                ++newAllocated;
            }
        }
        Allocated = newAllocated;
    }

    // This compares pointers, not strings!
    void ReleaseBuffer(char *knownCleared)
    {
        size_t newAllocated = 0;
        for (size_t i = 0; i < Allocated; ++i)
        {
            if (Pool[i].buffer == knownCleared)
            {
                delete[] Pool[i].buffer;
            }
            else
            {
                Pool[newAllocated] = Pool[i];
                ++newAllocated;
            }
        }
        Allocated = newAllocated;
    }

};
```

Share

Improve this answer

Follow

edited Nov 10, 2008 at 17:39

answered Nov 10, 2008 at 16:57

Eclipse

45.5k ● 20 ● 116 ● 172

2

Since std::string is not an option, for whatever reason, have you looked into smart pointers? See boost

But I can only encourage you to use std::string.

Christian

Share Improve this answer Follow

answered Nov 10, 2008 at 16:41

Christian Henning

Working on a limited platform means that a lot of things are not available, unfortunatly including boost. – xan Nov 10, 2008 at 16:42

You can write a simple enough array guard yourself to make this exception safe and to have the memory freed properly. See Lou Franco's answer. – Todd Gamblin Nov 10, 2008 at 16:45

2

If you must use char*'s, then LoadAndSetupData() should explicitly document who owns the memory for the char* after the call. You can do one of two things:

1. **Copy the string.** This is probably the simplest thing. LoadAndSetupData copies the string into some internal buffer, and the caller is always responsible for the memory.
2. **Transfer ownership.** LoadAndSetupData() documents that it will be responsible for eventually freeing the memory for the char*. The caller doesn't need to worry about freeing the memory.

I generally prefer safe copying as in #1, because the allocator of the string is also responsible for freeing it. If you go with #2, the allocator has to remember NOT to free things, and memory management happens in two places, which I find harder to maintain. In either case, it's a matter of **explicitly documenting** the policy so that the caller knows what to expect.

If you go with #1, take a look at Lou Franco's answer to see how you might allocate a char[] in an exception-safe, sure to be freed way using a guard class. Note that you can't (safely) use std::auto_ptr for arrays.

Share
Improve this answer
Follow

edited Nov 10, 2008 at 17:03

answered Nov 10, 2008 at 16:52

Todd Gamblin
59.7k ● 15 ● 91 ● 96

2

Since you need nameToUse to still exist after the function, you are stuck using new, what I would do is return a pointer to it, so the caller can "delete" it at a later time when it is no longer needed.

```
char * LoadStuff()
{
    char* suffix = GetSuffix();
    char* nameToUse = new char[50];
    sprintf("%s%s",mBaseName,suffix);

    LoadAndSetupData(nameToUse);
    return nameToUse;
}
```

then:

```
char *name = LoadStuff();
// do whatever you need to do:
delete [] name;
```

Share

Improve this answer

Follow

edited Nov 10, 2008 at 17:07

answered Nov 10, 2008 at 16:41

Evan Teran

90.3k ● 32 ● 187 ● 243

1

There is no need to allocate on heap in this case. And always use snprintf:

```
char nameToUse[50];
snprintf(nameToUse, sizeof(nameToUse), "%s%s",mBaseName,suffix);
```

Share Improve this answer Follow

answered Nov 10, 2008 at 16:58

ididak

5,858 ● 1 ● 22 ● 21

He said it must exist after the function exits. – Evan Teran Nov 10, 2008 at 17:17

I still think this is the right idea. He just needs to allocate nameToUse as an array at a level of scope where it is only deleted beyond the point where it is useful. – orcmid Nov 10, 2008 at 19:00

0

Where exactly nameToUse is used beyond the scope of LoadStuff? If someone needs it after LoadStuff it needs to pass it, along with the responisbility for memory deallocation

If you would have done it in c# as you suggested

```
LoadAndSetupData(mBaseName + GetSuffix());
```

then nothing would reference LoadAndSetupData's parameter, therefore you can safely change it to

```
char nameToUse[50];
```

as Martin suggested.

Share Improve this answer Follow

answered Nov 10, 2008 at 16:42

Dani
4,437 4 31 37

0

You're going to have to manage the lifetime of the memory you allocate for nameToUse. Wrapping it up in a class such as std::string makes your life a bit simpler.

I guess this is a minor outrage, but since I can't think of any better solution to your problem, I'll point out another potential problem. You need to be very careful to check the size of the buffer you're writing into when copying or concatenating strings. Functions such as strcat, strcpy and sprintf can easily overwrite the end of their target buffers, leading to spurious runtime errors and security vulnerabilities.

Apologies, my own experience is mostly on the Windows platform, where they introduced "safe" versions of these functions, called strcat_s, strcpy_s, and sprintf_s. The same goes for all their many related functions.

Share Improve this answer Follow

answered Nov 10, 2008 at 16:46

Martin
5,452 31 40

0

First: Why do you need for the allocated string to persist beyond the end of LoadStuff()? Is there a way you can refactor to remove that requirement.

Since C++ doesn't provide a straightforward way to do this kind of stuff, most programming environments use a set of guidelines about pointers to prevent delete/free problems. Since things can only be allocated/freed once, it needs to be very clear who "owns" the pointer. Some sample guidelines:

1) Usually the person that allocates the string is the owner, and is also responsible for freeing the string.

2) If you need to free in a different function/class than you allocated in, there must be an explicit hand-off of ownership to another class/function.

3) Unless explicitly stated otherwise, pointers (including strings) belong to the caller. A function, constructor, etc. cannot assume that the string pointer it gets will persist beyond the end of the function call. If they need a persistent copy of the pointer, they should make a local copy with strdup().

What this boils down to in your specific case is that LoadStuff() should delete[] nameToUse, and the function that it calls should make a local copy.

One alternate solution: if nameToUse is going to be passed lots of places and needs to persist for the lifetime of the program, you could make it a global variable. (This

saves the trouble of making lots of copies of it.) If you don't want to pollute your global namespace, you could just declare it static local to the function:

```
static char *nameToUse = new char[50];
```

Share

Improve this answer

Follow

edited Nov 10, 2008 at 16:57

answered Nov 10, 2008 at 16:49

JSBձոգչ

41.3k ● 19 ● 105 ● 171

0

Thankyou everyone for your answers. I have not selected one as "the answer" as there isn't a concrete solution to this problem and the best discussions on it are all upvoted be me and others anyway.

Your suggestions are all good, and you have been very patient with the clunkiness of my question. As I am sure you can see, this is a simplification of a more complicated problem and there is a lot more going on which is connected with the example I gave, hence the way that bits of it may not have entirely made sense.

For your interest I have decided to "cheat" my way out of the difficulty for now. I said that the base names were arbitrary, but this isn't quite true. In fact they are a limited set of names too, just a limited set that could change at some point, so I was attempting to solve a more general problem.

For now I will extend the "static" solution to suffixes and build a table of possible names. This is very "hacky", but will work and moreover avoids refactoring a large amount of complex code which I am not able to.

Feedback has been fantastic, many thanks.

Share Improve this answer Follow

answered Nov 10, 2008 at 17:35

xan

7,568 ● 8 ● 46 ● 70

0

You can combine some of the ideas here.

Depending on how you have modularized your application, there may be a method (main?) whose execution determines the scope in which nameToUse is definable as a fixed size local variable. You can pass the pointer (&nameToUse[0] or simply nameToUse) to those other methods that need to fill it (so pass the size too) or use it, knowing that the storage will disappear when the function having the local variable exits or your program terminates by any other means.

There is little difference between this and using dynamic allocation and deletion (since the pointer holding the location will have to be managed more-or-less the same way). The local allocation is more direct in many cases and is very inexpensive when there is no problem with associating the maximum-required lifetime with the duration of a particular function's execution.

Share Improve this answer Follow

answered Nov 10, 2008 at 19:09

orcmid

2,638 ● 19 ● 21

-1

I'm not totally clear on where LoadAndSetupData is defined, but it looks like it's keeping its own copy of the string. So then you should delete your locally allocated copy after the call to LoadAndSetupData and let it manage its own copy.

Or, make sure LoadAndSetupData cleans up the allocated char[] that you give it.

My preference would be to let the other function keep its own copy and manage it so that you don't allocate an object for another class.

Edit: since you use new with a fixed size [50], you might as well make it local as has been suggested and the let LoadAndSetupData make its own copy.

Share Improve this answer Follow

answered Nov 10, 2008 at 16:41

Jonathan Adelson

3,321 ● 5 ● 32 ● 39