How do you know how to deallocate an array of pointers?

Asked 9 years, 8 months ago Modified 9 years, 8 months ago Viewed 109 times

2

If I have an array of pointers like this:

```
char* p[3];

p[0] = new char;
p[1] = new char[10];
p[2] = &c;
```

Assuming I cannot use `std::string`, how would I know how to deallocate this without seeing the definition? How would I know to use `delete` or `delete[]` while iterating through the array, or whether it points to a stack variable or on the heap?

`c++`

Share Improve this question Follow

asked Apr 9, 2015 at 22:44

Me myself and I

4,060 ● 1 ● 25 ● 48

5 You wouldn't, so you would wrap them in suitable objects that manage it for you. – molbdnilo Apr 9, 2015 at 22:46

4 You wouldn't, which is why it's a horrendous idea to write code like that. – Kerrek SB Apr 9, 2015 at 22:48

You don't. A pointer stores an address and no other information, which is why you should never try to use them on their own to manage allocated memory. Always wrap them in other types (like `std::string`, if you don't have a mysterious reason not to use it) that know how they should be managed. – Mike Seymour Apr 9, 2015 at 22:51

and probably `delete p[2]` will be extremely fun (undefined behaviour), since I guess `c` is a stack/static variable – vsoftco Apr 9, 2015 at 22:56

If you cannot use `std::string` then the smart thing to do is immediately implement your own string class. – Neil Kirk Apr 9, 2015 at 23:00

4 Answers

Sorted by: Highest score (default)

The compiler does not save this information anywhere for you. You must save it yourself.

7

Share Improve this answer Follow

answered Apr 9, 2015 at 22:46

Bill Lynch

81.8k ● 16 ● 131 ● 179

4

Short answer: You don't, unless you know how it was allocated by the way the code is written.

Long answer: There is no (generic, portable way) to determine how or if something was allocated as an individual element with `new char` or as an array with `new char[10];` or not allocated at all. In theory, you could check if some address is "within the heap" if you know where the heap is, but there is no simple way to know what is heap, what is stack and what is global data without fairly intimate knowledge of the memory layout of that particular system, and compile the same code on a different OS or even different processor architecture of the same OS, and it all changes. To find out if it's a single or array allocation is even harder, if at all possible [most C++ runtime will not even detect this and complain when you do `char *p = new char[10]; delete p;` - it will just crash/misbehave or "work anyway, because it didn't matter", depending on your luck, C++ runtime library design and machine architecture] - see also further discussion below.

So you have to track that as part of your code [or not write code like that at all, which is my preference], or use some other construct (smart pointers would work, vectors would work).

Further: If you have a method for finding out whether something came from the heap or not, you still can't determine if it's "the original allocation or something else". Imagine the following:

```
char *p[2];

p[0] = new char[10];
p[1] = p[0] + 3;
```

Now, `p[1]` points inside the heap, but not at it's own allocation, but at a location within the allocation made by `p[0]`. So, basically, it's near impossible to do this, EVEN if we know where the heap, data and stack memory is located - which we can't know generically.

As a side note, people often say "the heap" as if it's a single contiguous piece of memory. It isn't in most modern OS's, because there are many different ways that a particular piece of memory may be occupied. It can be allocated as part of the code, data or stack loaded by the initial loading of your executable file. But it can also be part of a shared library (.so or .dll, etc) [which has code and data space] - and they are often given a specific address to avoid having to 'relocate' the shared library for every user, and a piece of memory could be a memory mapped file or shared memory allocation - which, at least sometimes, can be given a specific address in memory, and thus have an address "in the middle of the 'heap' memory region". So when we say "the heap", we really mean "any free memory address that the OS thinks we can use for storing things in", rather than one straight line of addresses from A to B with no holes. It's more like A-B, F-J, M, P and T-V that are "the heap".

And as Marcus mentions in the comment, there are OS's that intentionally "move things around" (address space randomization) to make it harder for someone with illicit purposes to rely on the distance from one memory region to another to abuse stack overwriting to "crack" the system.

Share
Improve this answer
Follow

edited Apr 9, 2015 at 23:09

answered Apr 9, 2015 at 22:50

Mats Petersson
129k ● 14 ● 146 ● 232

" There is no (generic, portable way)" There's even address space randomization, to make *exactly* that impossible. – Marcus Müller Apr 9, 2015 at 22:58

0

If you know you have an array that was created by `new[]`, you just delete it with `delete[]`; that's the contract you'll have to fulfill.

If you don't know whether something was allocated by you or not, you have what we call a memory leak, because you won't be able to free it, unless you want to risk crashing your program.

Share Improve this answer Follow

answered Apr 9, 2015 at 22:47

Marcus Müller
36.2k ● 4 ● 57 ● 100

0

I am thinking about it and it is a really good question, but I think, you just cannot.

As I am more used to programming C, I think it is impossible, as this information would have to be stored somewhere which (at least in C) is not the case, as far as I know.

Share Improve this answer Follow

answered Apr 9, 2015 at 22:47

benaryorg

1,097 1 10 22