

What are the often misunderstood concepts in C++? [closed]

Asked 15 years, 10 months ago Modified 12 years, 5 months ago

Viewed 7k times



33



As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](#) for guidance.

Closed 12 years ago.

What are the often misunderstood concepts in c++?

c++

Share

edited Feb 18, 2009 at 12:50

Improve this question

Follow

community wiki

3 revs, 3 users 100%

yesraaj

similar to: stackoverflow.com/questions/75538/hidden-features-of-c – Mitch Wheat Feb 18, 2009 at 12:31

more like [stackoverflow.com/questions/294018/...](https://stackoverflow.com/questions/294018/)
– Tim Matthews Feb 18, 2009 at 12:32

1 also similar to: [stackoverflow.com/questions/174892/...](https://stackoverflow.com/questions/174892/)
– Scott Langham Feb 18, 2009 at 12:34

38 Answers

Sorted by:

Highest score (default)



1

2

Next



C++ is not C with classes!

80

And there is no language called C/C++. Everything goes downhill from there.



Share Improve this answer

answered Feb 18, 2009 at 12:43



Follow



community wiki


[Konrad Rudolph](#)




4 haha, too true. I like the one about C/C++. :)
– [Stack Overflow is garbage](#) Feb 19, 2009 at 0:21

4 There may not be C/C++, but I'm sure there is C/C++/C#!
– [MiseryIndex](#) Nov 23, 2009 at 21:46

C/C++ most certainly does exist; it's the concept of writing programs which you can compile as C but when compiled as C++ adds features or checking. This means you can make a

codebase whose footprint does not *require* a C++ compiler, but benefits from it when available. Furthermore, C++'s whole raison d'être was to permit mixtures of C and C++ code for gradual improvements--a fact people often forget. Of course, most people asking questions tagged both `c` and `c++` are not dealing with either scenario. But suggesting the discussion of hybridized C and C++ code is not legitimate is wrong. – [HostileFork says dont trust SE](#) Nov 18, 2018 at 10:39 

@HostileFork Nobody is suggesting this. I am saying C/C++ *the language* does not exist. Anyway, C++'s original design might have been intentionally such that code bases could be gradually lifted from C but this is no longer the case. A lot of modern, valid C code is simply not valid C++ and that's OK. And C++ nowadays is no longer the same language as back in the days, so don't confuse history with what's currently true. – [Konrad Rudolph](#) Nov 19, 2018 at 11:13 



40



That C++ *does* have automatic resource management.

(Most people who claim that C++ does not have memory management try to use new and delete way too much, not realising that if they allowed C++ to manage the resource themselves, the task gets much easier).



Example: (Made with a made up API because I do not have time to check the docs now)

```
// C++
void DoSomething()
{
    File file("/tmp/dosomething", "rb");
    ... do stuff with file...
    // file is automatically free'd and closed.
```

```

}

// C#
public void DoSomething()
{
    File file = new File("/tmp/dosomething", "rb");
    ... do stuff with file...

    // file is NOT automatically closed.
    // What if the caller calls DoSomething() in a tight
    // C# requires you to be aware of the implementation
    // and forces you to accommodate, thus voiding imple
    // principles.
    // Approaches may include:
    // 1) Utilizing the IDisposable pattern.
    // 2) Utilizing try-finally guards, which quickly ge
    // 3) The nagging doubt that you've forgotten someth
    //     1 million loc project.
    // 4) The realization that point #3 can not be fixed
    //     class.
}

```

Share Improve this answer

edited Feb 22, 2009 at 23:47

Follow

community wiki

2 revs

Arafangion

-1 please explain how and where doe sit have automatic resource management? – [hasen](#) Feb 19, 2009 at 14:22

- 4 It is ironic that the most correct answers to this question might be modded down the most, *because* they are more often misunderstood. (That said, I should've mentioned several keywords to help people google the reference, such as RAI and the preference to allocate objects on the stack.) – [Arafangion](#) Feb 19, 2009 at 22:39
-

Well, just edit the post, and justify the answer. imho, C++ really does *not* have automatic resource management, even if its proponents say so. – [hasen](#) Feb 19, 2009 at 23:29

- 3 @hasen j: C++ have an *optional* automatic resource management. In C++ the `auto` keyword is the default storage class for (stack-allocated) local variables. Heap-allocation, as marked by using the `new` keyword, are used to mark a resource for manual resource management. However, all too often, a programmer misused the `new` keyword and caused memory leaks; then they claim C++ does not have automatic resource management, when they just have disabled it. – [Lie Ryan](#) Sep 27, 2010 at 14:31
-

All he is saying is: if you allocate a class on the stack, then when it goes out of scope, the object is destroyed and it's destructor called. It is a useful technique but I don't know that it is worthy of being called automatic memory management because one may have a good reason to allocate on the heap. – user140327 Jan 9, 2011 at 4:16



35

Free functions are not bad just because they are not within a class C++ is not an OOP language alone, but builds upon a whole stack of techniques.



I've heard it many times when people say free functions (those in namespaces and global namespace) are a



"relict of C times" and should be avoided. Quite the



opposite is true. Free functions allow to decouple functions from specific classes and allow reuse of functionality. It's also recommended to use free functions instead of member functions if the function don't need access to implementation details - because this will

eliminate cascading changes when one changes the implementation of a class among other advantages.

This is also reflected in the language: The range-based for loop in `C++0x` (next C++ version released very soon) will be based on free function calls. It will get begin / end iterators by calling the free functions `begin` and `end`.

Share Improve this answer

edited Aug 26, 2009 at 0:40

Follow

community wiki

5 revs

Johannes Schaub - litb

hey, weren't you arguing for member functions a week or two ago? ;) I agree though. And grats on your gold badge! By the way, isn't it still C++0x? Last I heard, they were still aiming to finalize the spec in 09. – [Stack Overflow is garbage](#) Feb 19, 2009 at 0:30

jalf, well herb sutter said at oct'08 that there will be a second community draft soon at around october'09. then i think they will await comments again and have a meeting, after which finally the new standard will come out in 2010. that's what i heard and why i call it c++1x :) – [Johannes Schaub - litb](#) Feb 19, 2009 at 2:54

-
- 1 I absolutely agree. Developers seem to put everything in a class, whether it belongs there or not. also I prefer c++0A instead of c++1X. – [deft_code](#) Feb 21, 2009 at 15:48
-

- 1 It's called Interface Principle and it's the idea of Herb Sutter. It would be nice to give him some credit at least.
gotw.ca/publications/mill08.htm – Piotr Dobrogost Jun 7, 2009 at 20:58
- 4 @Piotr, it was not Herb Sutters' article that convinced me that free functions are superior, but it was Scott Meyers' "How non-member functions improve encapsulation" that convinced me: ddj.com/cpp/184401197 Without having looked up who has had the most influence on this, I don't think this is something you can attribute to some single person. Rather, the mechanism and ideas have grown up in the community as a whole, and all members of it somehow attribute to the things that come out of it.
– Johannes Schaub - litb Jun 9, 2009 at 0:19



27



The difference between assignment and initialisation:

```
string s = "foo";    // initialisation
s = "bar";           // assignment
```

Initialisation always uses constructors, assignment always uses operator=

Share Improve this answer

answered Feb 18, 2009 at 12:52

Follow

community wiki
[anon](#)

- 8 I always do initialization with parenthesis (string s("foo");), to help keep the difference obvious. The difference between init

and assignment inside a constructor is also misunderstood.

– [Chris Smith](#) Feb 18, 2009 at 14:33

How is this a "misunderstood concept" ? – [Nippysaurus](#) Jul 27, 2009 at 23:31

Another example might be `Object o(1);` vs `Object o = 1;` where `Object` has both a copy constructor and a single argument constructor that takes an `int`. – [James Schek](#) Jul 28, 2009 at 15:18



In decreasing order:

22



1. make sure to release pointers for allocated memory
2. *when destructors should be virtual*
3. how virtual functions work



Interestingly not many people know the full details of virtual functions, but still seem to be ok with getting work done.

Share Improve this answer

[edited Feb 18, 2009 at 14:12](#)

Follow

community wiki

[2 revs](#)

[Sesh](#)

- 2 1. Use smart pointers for allocated memory. It'll make your life so much easier. – [David Thornley](#) Feb 18, 2009 at 14:58
-

@David - you are right. However the question is about misunderstood concepts in C++, and memory management

tops the list at least for what I have seen. – [Sesh](#) Feb 18, 2009 at 15:13

Guilty of #2. I think. =[+1 anyway. – [strager](#) Feb 18, 2009 at 23:28

Actually, once you know 3., 2. becomes easy. But you have to know what vtables are. – [isekaijin](#) Feb 19, 2009 at 7:54

@Sesh: How about "1. Use smart pointers, or make sure to delete raw pointers to allocated memory."? – [David Thornley](#) Nov 23, 2009 at 21:52



21



The most pernicious concept I've seen is that it should be treated as C with some addons. In fact, with modern C++ systems, it should be treated as a different language, and most of the C++-bashing I see is based on the "C with add-ons" model.



To mention some issues:



While you probably need to know the difference between `delete` and `delete[]`, you should normally be writing neither. Use smart pointers and `std::vector<>`.

In fact, you should be using a `*` only rarely. Use `std::string` for strings. (Yes, it's badly designed. Use it anyway.)

RAII means you don't generally have to write clean-up code. Clean-up code is bad style, and destroys conceptual locality. As a bonus, using RAII (including smart pointers) gives you a lot of basic exception safety

for free. Overall, it's much better than garbage collection in some ways.

In general, class data members shouldn't be directly visible, either by being `public` or by having getters and setters. There are exceptions (such as `x` and `y` in a point class), but they are exceptions, and should be considered as such.

And the big one: there is no such language as C/C++. It is possible to write programs that can compile properly under either language, but such programs are not good C++ and are not normally good C. The languages have been diverging since Stroustrup started working on "C with Classes", and are less similar now than ever. Using "C/C++" as a language name is *prima facie* evidence that the user doesn't know what he or she is talking about. C++, properly used, is no more like C than Java or C# are.

Share Improve this answer

answered Feb 18, 2009 at 15:21

Follow

community wiki

[David Thornley](#)

A C program that compiles as C++ is not normally good C?

Nearly all C programs compile as C++, in my experience.

– [Imbue](#) Feb 19, 2009 at 8:09

Depends on what it does. "`int * a = (int *)malloc(...)`" isn't really good C, and "`int * a = malloc(...)`" doesn't compile under C++. Wherever there's compromises, the C++ side is usually

not as good C as the C side. – [David Thornley](#) Feb 19, 2009 at 14:23

- 3 `std::string` isn't badly designed -- it just doesn't have most of it's functionality as part of it's signature. Use STL style algorithms on it instead. Too many C++ programmers look at `std::string` as a crappy version of a Java or C# string, rather than as a full fledged STL container.
– [Billy ONeal](#) Sep 24, 2010 at 17:37
-

- 1 AFAICT C++ is an open ended logical system via template programming. C is a closed system. In C, A is A, but in C++ A might be `AA<T>`. I don't know if this makes sense to you, but it does to me. – [user140327](#) May 19, 2011 at 5:14
-

@BillyONeal yes and add over that the very necessary boost string algorithms and it closes in on python or C#. and in C++11 there even are facets now to do some utf encoding. cf `codecvt` header – [v.oddou](#) Mar 4, 2015 at 4:59



19



The overuse of inheritance unrelated to polymorphism. Most of the time, unless you really do use runtime polymorphism, composition or static polymorphism (i.e., templates) is better.

Share Improve this answer

answered [Feb 18, 2009 at 13:48](#)



Follow



community wiki
[KeithB](#)

- 2 Nice one! Though maybe it's not limited to the C++ users.
– [xtofi](#) Feb 18, 2009 at 14:23
-

NOT limited to C++ for sure, although the use of templates gives more options than Java, for instance. – [KeithB](#) Feb 18, 2009 at 14:25

Then again, Java does have generics now, but suffers from having to box/unbox primitive types... – [Arafangion](#) Feb 19, 2009 at 2:40

- 4 Java generics aren't a replacement for the template system in C++. See this question:
stackoverflow.com/questions/498317/... – [KeithB](#) Feb 19, 2009 at 13:09
-

This answer was related to *polymorphism*. – [Arafangion](#) Mar 5, 2009 at 6:42



The **static** keyword which can mean one of three distinct things depending on where it is used.

13



1. It can be a static member function or member variable.
2. It can be a static variable or function declared at namespace scope.
3. It can be a static variable declared inside a function.



Share Improve this answer

[edited Feb 18, 2009 at 12:50](#)

Follow

community wiki

[2 revs](#)

[sharptooth](#)

I can only remember two of those three things: declaring a static method and declaring a static variable, what's the third one? – [isekaijin](#) Feb 18, 2009 at 12:48

I meant, a static member, not necessarily a method.
– [isekaijin](#) Feb 18, 2009 at 12:49

Note that usage #2 is deprecated. Unnamed namespaces are preferred to the static keyword for this use case.
– [Brian Neal](#) Feb 18, 2009 at 14:30

What am I missing? If unnamed namespaces can replace #2 then how do you distinguish between 2 identically named functions/variables that in the past would have different namespaces to delineate between them? – [Dunk](#) Feb 18, 2009 at 15:08

Dunk: What do you mean? If they are in an unnamed namespace, they're not visible outside the current compilation unit, so they won't conflict with identically named symbols in other compilation units. And if they're in the same compilation unit, it's a violation of ODR, and illegal.
– [Stack Overflow is garbage](#) Feb 19, 2009 at 0:24



Arrays are not pointers

13



They are different. So `&array` is not a pointer to a pointer, but a pointer to an array. This is the most misunderstood concept in both C and C++ in my opinion. You gotta have a visit to all those SO answers that tell to pass 2-d arrays as `type**` !



Share Improve this answer

[edited Feb 18, 2009 at 23:26](#)

Follow

community wiki
2 revs
[Johannes Schaub - litb](#)



Here is an important concept in C++ that is often forgotten:

12



C++ should not be simply used like an object oriented language such as Java or C#. Inspire yourself from the STL and write generic code.



Share Improve this answer

[edited Feb 18, 2009 at 13:27](#)

Follow

community wiki
2 revs
[Edouard A.](#)

Why not? It was primarily designed to support the object oriented programming paradigm. – [Scott Langham](#) Feb 18, 2009 at 12:50

- 1 Because it has no garbage collection. Also, because virtual functions are messed up. Also, it has no strings, so each library ships with its own string class. – [hasen](#) Feb 18, 2009 at 12:56

Because it's more generic oriented than object oriented. See the STL. – [Edouard A.](#) Feb 18, 2009 at 13:19

C++ supports OO fine, the difference is that you have to do a lot of headachy work compared to, say C#. This doesn't

mean you shouldn't use it for OO... just that its more difficult to do it safely. At least imho – [jheriko](#) Feb 18, 2009 at 13:24

- 3 It is better if you restate your assertion: "C++ is much more than OO, if you look at it just as an OO language, you are missing important features like generic programming".
– [Ismael](#) Feb 18, 2009 at 14:05
-



12



Here are some:

1. Using templates to implement polymorphism without vtables, à la ATL.
2. Logical `const`-ness vs actual `const`-ness in memory. When to use the `mutable` keyword.

ACKNOWLEDGEMENT: Thanks for correcting my mistake, spoulson.

EDIT:

Here are more:

1. Virtual inheritance (not virtual methods): In fact, I don't understand it at all! (by that, I mean I don't know how it's implemented)
2. Unions whose members are objects whose respective classes have non-trivial constructors.

Share Improve this answer

[edited Feb 19, 2009 at 7:47](#)

Follow

Typo : "mutable" instead of "mutabile" – [Klaim](#) Feb 18, 2009 at 13:09

I don't understand point 1 – or rather, I understand it, but I don't agree with it, at all. – [Konrad Rudolph](#) Feb 18, 2009 at 14:52

i agree with 1., but i don't see how it is a misunderstood concept? it's rather a feature of c++ than a misunderstood thing. can u please add what's misunderstood there?
– [Johannes Schaub - litb](#) Feb 18, 2009 at 23:21

@Konrad: If you don't agree with point 1, then at least you or the author misunderstood it. That's a 50% misunderstanding. And that's often enough for it to be a good answer to this question. [Maybe my logic's a bit perverse... I think I need sleep]. – [Scott Langham](#) Feb 18, 2009 at 23:23

Scott i thought about the same, lol. If he says it's a valid point, and Konrad says it isn't, one of them didn't get it :p
– [Johannes Schaub - litb](#) Feb 18, 2009 at 23:27



Given this:

12

```
int x = sizeof(char);
```



what value is X?



The answer you often hear is dependant on the level of understanding of the specification.



1. Beginner - x is one because chars are always eight bit values.
2. Intermediate - it depends on the compiler implementation, chars could be UTF16 format.
3. Expert - x is one and always will be one since a char is the smallest addressable unit of memory and sizeof determines the number of units of memory required to store an instance of the type. So in a system where a char is eight bits, a 32 bit value will have a sizeof of 4; but in a system where a char is 16 bits, a 32 bit value will have a sizeof of 2.

It's unfortunate that the standard uses 'byte' to refer to a unit of memory since many programmers think of 'byte' as being eight bits.

Share Improve this answer

edited Jul 3, 2012 at 14:02

Follow

community wiki

2 revs, 2 users 93%

Skizz

5 Your answer for '3' is wrong... It is true that x is will always be one - but your rationale is wrong! It is defined as one by the spec, not because it is the smallest addressable unit of memory. Some archs probably address bits, while others can't address less than 32 bits - x is still one! – [Arafangion](#)
Feb 19, 2009 at 0:00


1 OK, perhaps my wording wasn't great there. However, it's the compiler writer that decides the size of a byte/char, not the

hardware. On an IA32, a byte could be 16 bits even though the hardware can address to 8 bit resolution. Obviously, the compiler writer would choose a size that was most efficient

– [Skizz](#) Feb 19, 2009 at 0:36

You're worrding is weird. You say x is always one, and yet sizeof(char) will return either 2 or 4? I don't grok? – [Chris K](#) Jul 27, 2009 at 23:30

It's even more unfortunate that "char" refers to a unit of memory now that we have Unicode. – [dan04](#) Mar 12, 2010 at 5:22

- 1 The "beginner" sounds like the expert among them (being a little forgiving, since as an expert, he should have said that `sizeof(char)` is defined by the language standard as of C99 to be one *byte*). The "expert" sounds like the beginner, for both making no sense, being confusing, and ultimately wrong. The "intermediate" seems also a bit confused, but at least isn't spouting nonsense with over-confidence, as `sizeof` other data types do in fact depend on architecture and/or compiler. Pointing out that `sizeof(char)=1` is the exception should be simple enough. – [swalog](#) Feb 15, 2015 at 20:51 
-



a classic among beginners to c++ from c:

10

confuse `delete` and `delete[]`



EDIT:



another classic failure among all levels of experience when using C API:



```
std::string helloString = "hello world";
```

```
printf("%s\n", helloString);
```

instead of:

```
printf("%s\n", helloString.c_str());
```

it happens to me every week. You could use streams, but sometimes you have to deal with printf-like APIs.

Share Improve this answer

edited Nov 23, 2009 at 21:38

Follow

community wiki
3 revs, 2 users 94%
davidnr

Looks like that beginner is confusing arrays with `std::Vector`.
;) – [Arafangion](#) Feb 19, 2009 at 2:41

I think beginners usually don't use the `stl`, just use C++ as C with classes. – [davidnr](#) Feb 19, 2009 at 15:51

`printf("%s\n", helloString);` won't even compile, I predict.
– [mannicken](#) Feb 19, 2009 at 18:10

visual c++ allows as an extension to pass non-pod classes to var-arg functions, i read. for example passing a `CString` directly. but i forgot the specific semantics
– [Johannes Schaub - litb](#) Feb 19, 2009 at 19:32

-
- 1 @mannicken: According to the standard 5.2.2.7: "If the argument [matching an ellipsis] has non-POD class type, the behaviour is undefined." – [j_random_hacker](#) Feb 23, 2009 at 0:56
-



C++ is a multi-paradigm language. Many people associate C++ strictly with OOP.

9

Share Improve this answer

edited Feb 18, 2009 at 23:45



Follow



community wiki

2 revs

cdv



Pointers.

9

Dereferencing the pointers. Through either `.` or `->`



Address of using `&` for when a pointer is required.

Functions that take params by reference by specifying a `&` in the signature.



Pointer to pointers to pointers `***` or pointers by reference `void someFunc(int *& arg)`

Share Improve this answer

edited Feb 19, 2009 at 1:22

Follow

community wiki

4 revs, 2 users 83%

Tim Matthews

Sorry couldn't figure out SO's escape characters so may appear weird. – [Tim Matthews](#) Feb 18, 2009 at 12:37

Here you go. =] Backticks (`) surround inline code. – [strager](#) Feb 18, 2009 at 23:29

You can't have pointers to references, they're illegal. You can have references to pointers, though. – [Adam Rosenfield](#) Feb 18, 2009 at 23:31

@Adam Rosenfield sorry it was about 2am when I posted this. I have rearranged the last example. – [Tim Matthews](#) Feb 19, 2009 at 1:23



There are a few things that people seem to be constantly confused by or have no idea about:

9



1. Pointers, especially function pointers and multiple pointers (e.g. `int(*) (void*)`, `void***`)
2. The `const` keyword and `const` correctness (e.g. what is the difference between `const char*`, `char* const` and `const char* const`, and what does `void class::member() const`; mean?)
3. Memory allocation (e.g. every pointer new'ed should be deleted, `malloc/free` should not be mixed with `new/delete`, when to use `delete []` instead of `delete`, why the C functions are still useful (e.g. `expand()`, `realloc()`))
4. Scope (i.e. that you can use `{ }` on its own to create a new scope for variable names, rather than just as part of `if`, for etc...)



5. Switch statements. (e.g. not understanding that they can optimise as well (or better in some cases) than chains of ifs, not understanding fall through and its practical applications (loop unrolling as an example) or that there is a default case)
6. Calling conventions (e.g. what is the difference between cdecl and stdcall, how would you implement a pascal function, why does it even matter?)
7. Inheritance and multiple inheritance and, more generally, the entire OO paradigm.
8. Inline assembler, as it is usually implemented, is not part of C++.

Share Improve this answer

edited Feb 19, 2009 at 15:36

Follow

community wiki

[2 revs](#)

[jheriko](#)

1 While #6 may be important, it has nothing to do with standard C++. Rather it is an extension for a specific OS. – [KeithB](#) Feb 18, 2009 at 13:46

1 Not to mention that #1, #2, #4, #5, and #8 apply just as much to C. – [David Thornley](#) Feb 18, 2009 at 15:03

It is true that calling conventions are implementation specific, but I feel it is still important to understand their usage and why they may prevent code from functioning if not dealt with correctly since every implementation of C++ depends on them. Thanks for the feedback though. :) – [jheriko](#) Feb 18, 2009 at 18:48

In-line assembler in the shape of the `asm()` declaration is part of Standard C++ (7.4/1) – anon Feb 18, 2009 at 22:38

Isn't the language "assembly"? "Assembler" is the tool, like "compiler." =] – [strager](#) Feb 18, 2009 at 23:33



8



- Pointers to members and pointers to member functions.
- Non-type template parameters.
- Multiple inheritance, particularly virtual base classes and shared base objects.
- Order of construction and destruction, the state of virtual functions in the middle of constructing an intermediate base class.
- Cast safety and variable sizes. No, you can't assume that `sizeof(void *) == sizeof(int)` (or any other type for that matter, unless a portable header specifically guarantees it) in portable code.
- Pointer arithmetic.

Share Improve this answer

answered [Feb 18, 2009 at 14:19](#)

Follow

I've gotten caught by your fourth bullet once or twice before.
>_< Note: [u]intptr_t <stdint.h> must at least be the size of a pointer. – [strager](#) Feb 18, 2009 at 23:36

A nice, tight collection of often-confused aspects of C++.
– [j_random_hacker](#) Feb 23, 2009 at 1:00



Headers and implementation files

7



This is also a concept misunderstood by many. Questions like what goes into header files and why it causes link errors if function definitions appear multiple times in a program on the one side but not when class definitions appear multiple times on the other side.



Very similar to those questions is why it is important to have *header guards*.

Share Improve this answer

answered [Feb 19, 2009 at 17:35](#)

Follow

community wiki
[Johannes Schaub - litb](#)



If a function accepts a pointer to a pointer, `void*` will still do it

7



I've seen that the concept of a *void pointer* is frequently confused. It's believed that if you have a pointer, you use a `void*`, and if you have a pointer to a pointer, you use a `void**`. But you can and should in both cases use `void*`. A `void**` does not have the special properties that a `void*` has.

It's the special property that a `void*` can also be assigned a pointer to a pointer and when cast back the original value is received.

Share Improve this answer

edited Sep 24, 2010 at 17:29

Follow

community wiki

2 revs

Johannes Schaub - litb

an example use case? `void f(void *p) { int **a = (int**)p; *a = some_int_ptr; }` instead of `void f(void **p);` – Johannes Schaub - litb Aug 26, 2009 at 0:19



5



`NULL` is always zero.

Many confuse `NULL` with an address, and think therefor it's not necessarily zero if the platform has a different null pointer address.

But `NULL` is always zero and it is not an address. It's an zero constant integer expression that can be converted to



pointer types.

Share Improve this answer

answered [Aug 26, 2009 at 0:21](#)

Follow

community wiki

[Johannes Schaub - litb](#)



Memory Alignment.

4

Share Improve this answer

answered [Feb 18, 2009 at 12:41](#)

Follow



community wiki

[Tim Matthews](#)



I don't think this is a C++ concept per se. – [Ismael](#) Feb 18, 2009 at 13:52

+1 because I don't even know what it means or why it matters, but I keep hearing about it – [Iraimbilanja](#) Apr 16, 2009 at 21:04



`std::vector` does not create elements when reserve is used

4

I've seen it that programmers argue that they can access members at positions greater than what `size()` returns if they `reserve()` 'ed up to that positions. That's a wrong





assumption but is very common among programmers - especially because it's quite hard for the compiler to diagnose a mistake, which will silently make things "work".

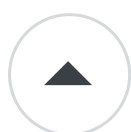
Share Improve this answer

answered [Feb 19, 2009 at 17:40](#)

Follow

community wiki
[Johannes Schaub - litb](#)

It just reserves memory, not construct objects to fill the unused space. – user140327 May 19, 2011 at 4:56



4



I think the most misunderstood concept about C++ is why it exists and what its purpose is. Its often under fire from above (Java, C# etc.) and from below (C). C++ has the ability to operate close to the machine to deal with computational complexity and abstraction mechanisms to manage domain complexity.



Share Improve this answer

answered [Mar 5, 2009 at 12:06](#)

Follow

community wiki
[Nikhil](#)



Hehe, this is a silly reply: the most misunderstood thing in C++ programming is the error messages from g++ when

4 template classes fail to compile!



Share Improve this answer

answered [Jan 9, 2011 at 5:59](#)

Follow



community wiki
[user140327](#)



C++ is not C with string and vector!

3

Share Improve this answer

answered [Feb 18, 2009 at 14:25](#)

Follow



community wiki
[Igor](#)

Ha! A lot of people just use it like C, *without* using string or vector. :) I never want to see a strcmp or strcpy in C++ code again! – [Brian Neal](#) Feb 18, 2009 at 14:34

And what if you can't use the stl or boost because of platform restriction, but you need OO? You write your own string library (hopefully) and use strlen etc. internally, or you use char*'s and be done with it. – [xan](#) Feb 18, 2009 at 15:57

@xan - why can't you use STL? I can understand about not wanting or being able to run boost, but STL is standard with all major compiler vendors. It is a horrifically bad idea to write your own string library when you can use std::string (even with all its faults). – [Brian Neal](#) Feb 18, 2009 at 19:48

Using C++ like C is a valid C++ style. It was the goal of the language to build upon C, not remove it. – [Chris de Vries](#) Feb

18, 2009 at 23:33

cdv: No it isn't, and no it wasn't. – [Stack Overflow is garbage](#)
Feb 19, 2009 at 0:27



C structs VS C++ structs is often misunderstood.

3

Share Improve this answer

answered [Feb 19, 2009 at 1:26](#)

Follow



community wiki
[vdsf](#)



In what way? I understand that a C++ struct is identical to a class (except for default access specifier), but if you think of it as identical to a C struct, and use class for everything else, I think you end up with clearer code and no confusion.

– [KeithB](#) Feb 19, 2009 at 17:03

Just C++ struct can have member functions but C struct cannot. C structs are just data, C++ structs benefit from ctor/dtor and inheritance. Everything is public in either. Look at eg LIBJPEG, a C library, and what it's authors called "poor man's object oriented programming": C structs with C function pointer members assigned to whatever C function the particular JPEG file represented by a struct being en/decoded requires. – [user140327](#) May 19, 2011 at 5:08



C++ is not a typical object oriented language.

3

Don't believe me? look at the STL, way more templates than objects.



It's almost impossible to use Java/C# ways of writing object oriented code; it simply doesn't work.

- In Java/C# programming, there's a lot of `new` ing, lots of utility objects that implement some single cohesive functionality.
- In C++, any object `new` ed must be deleted, but there's always the problem of who owns the object
- As a result, objects tend to be created on the stack
- But when you do that, you have to copy them around all the time if you're going to pass them around to other functions/objects, thus wasting a lot of performance that is said to be achieved with the unmanaged environment of C++
- Upon realizing that, you have to think about other ways of organizing your code
- You might end up doing things the procedural way, or using metaprogramming idioms like smart pointers
- At this point, you've realized that OO in C++ cannot be used the same way as it is used in Java/C#

[Q.E.D.](#)

If you insist on doing oop with pointers, you'll usually have large (gigantic!) classes, with clearly defined ownership relationships between objects to avoid memory leaks.

And then even if you do that, you're already too far from the Java/C# idiom of oop.

Actually I made up the term "object-oriented", and I can tell you I did not have C++ in mind.

-- [Alan Kay](#) (click the link, it's a video, the quote is at 10:33)

Although from a purist point of view (e.g. Alan Kay), even Java and C# fall short of true oop

Share Improve this answer

edited Feb 19, 2009 at 8:14

Follow

community wiki

3 revs

[hasen j](#)

Which is more expensive - copying memory, or constantly dereferencing references? I'm not sure if I would agree that copying the objects wastes alot of performance gains... If the object is large, though, then optimize it in the class! (Many std::strings use a reference, internally, afaik). – [Arafangion](#) Feb 19, 2009 at 2:38

it doesn't matter which choice you follow, either way, you're already too far from the Java/C# idiom of oop. – [hasen](#) Feb 19, 2009 at 8:07

-
- 1 Arguably-what if you added a garbage collector, and only passed around smart pointers? Either way, it doesn't matter- you're abusing RAI, which is C++'s best feature, so you're right, even if someone somehow uses C# ways in C++,

you've used the worst of both worlds. – [Arafangion](#) Feb 19, 2009 at 22:43

Template meta-programming and object oriented programming are two different things. C++ supports both.
– user140327 May 19, 2011 at 5:00



A pointer is an iterator, but an iterator is not always a pointer

3



This is also an often misunderstood concept. A pointer to an object is a random access iterator: It can be incremented/decremented by an arbitrary amount of elements and can be read and written. However, an iterator class that has operator overloads doing that fulfill those requirements too. So it is also an iterator but is of course not a pointer.



I remember one of my past C++ teachers was teaching (wrongly) that you get a pointer to an element of a vector if you do `vec.begin()`. He was actually assuming - without knowing - that the vector implements its iterators using pointers.

Share Improve this answer

answered Feb 19, 2009 at 17:46

Follow

community wiki
[Johannes Schaub - litb](#)

-
- 1 Schaub: IMO iterator is conceptually a pointer in the sense that an iterator points to a specific object. I agree though, that iterator is not a pointer in the sense of a "memory address".
– [Lie Ryan](#) Sep 27, 2010 at 14:49
-



1



- That anonymous namespaces are almost always what is truly wanted when people are making static variables in C++
- When making library header files, the pimpl idiom (<http://www.gotw.ca/gotw/024.htm>) should be used for almost all private functions and members to aid in dependency management

Share Improve this answer

answered [Feb 18, 2009 at 14:46](#)

Follow

community wiki
[Spyplane](#)

I'll retort with "believing that anonymous namespaces and the `static` keyword are interchangeable". They get similar results, but using different means, and sometimes you want both. `static` variables don't show up outside of the compilation unit, whereas anonymous-namespaced variables do. That can make a huge difference in shared library name binding performance. – [Tom](#) Aug 26, 2009 at 0:45



I still don't get why vector doesn't have a `pop_front` and the fact that I can't `sort(list.begin(), list.end())`..

1

Share Improve this answer

answered Feb 23, 2009 at 0:39



Follow



community wiki
[user35978](#)

I think that's because both would be fairly inefficient. You could probably find lots of explanations of why that is on SO or elsewhere. If you really want to use `pop_front`, use a deque, and if you really want to sort something, it should be an array or vector (or a tree, or...) – [Tyler](#) Apr 4, 2009 at 5:54

1

2

Next