# When should you use a class vs a struct in C++?

In what scenarios is it better to use a `struct` vs a `class` in C++?

c++ oop class struct ooad

edited Aug 28 '16 at 15:34 Tamir Vered 8,249 ●4 ●34 ●47

asked Sep 10 '08 at 16:29 Alan Hinchcliffe 3,533 ●3 ●13 ●10

28 This isn't only applicable to C++, but to any language that provides both structs and classes. – Jason Bunting Sep 10 '08 at 16:31

2 I still disagree - I approach this question semantically. Maybe there are some technical differences, but semantically, they are not. Structs are really useful for creating value types, classes are not. – Jason Bunting Sep 10 '08 at 17:22

1 I find that I rarely use struct in C++, except when I need auto aggregate initialization of Plain Old Data, I don't think that can be done with data structures defined with an instance of a class. – Roger Nelson Sep 17 '08 at 5:06

1 I believe that there is no serious reason for using structs in C++. To me structs are another redundant "feature" of C++ that exists only for compatibility with C, like typedefs. These would not exist if C++ was not initially treated as an extension to C, and was designed from scratch, such as Java. In general, I find that many of the weirdest things about C++ have to do with C compatibility. – Kostas Jun 26 '09 at 11:31

5 Struct - For POD (plain old data) and all member accessibility is public. Class - When you need better encapsulation and need member functions to work with the class's state. – Navaneeth K N Jul 28 '09 at 3:51

## 23 Answers

Differences between a `class` and a `struct` in C++ are that structs have default `public` members and bases and classes have default `private` members and bases. Both classes and structs can have a mixture of `public` and `private` members, can use inheritance and can have member functions.

I would recommend using structs as plain-old-data structures without any class-like features, and using classes as aggregate data structures with `private` data and member functions.

edited Feb 7 '17 at 19:00 HolyBlackCat 10.7k ●2 ●27 ●50

answered Sep 10 '08 at 16:35 Commodore Jaeger 21k ●4 ●46 ●44

71 A struct with no modifiers or methods is called a POD struct, which exists as a backwards compatible interface with C libraries as it is (supposedly) guaranteed to be laid out as though it were a C struct. Apart from this one exception though, the only difference is as stated. – workmad3 Sep 18 '08 at 14:22

20 @workmad3: The name is misleading, but 9/4 (C++03) says: "A POD-struct is an aggregate class that has no non-static data members of type non-POD-struct, non-POD-union (or array of such types) or reference, and has no user-defined copy assignment operator and no user-defined destructor." There is no restriction on using the "struct" class-key and no restriction on using "public" (see 8.5.1/1 for aggregate requirements). This is not a difference between "struct" and "class". – Roger Pate Sep 22 '10 at 20:31

4 Your use of "aggregate" could be misunderstood, given the standard's definition. :) – Roger Pate Sep 22 '10 at 20:32

37 @wordmad3: *"Apart from this one exception though, the only difference is as stated."* Why does this total nonsense have 35 upvotes? A `class` with the same properties is POD too. Aside from the defaults imbued depending upon the keyword used at declaration, there is absolutely no semantic difference between "a struct" and "a class"; indeed, they are in fact called the same thing! You are defining a *user-defined type* nothing more nothing less. – Lightness Races in Orbit Feb 7 '15 at 4:50

3 This answer is *extremely* misleading. – Lightness Races in Orbit Apr 28 '16 at 14:12

As everyone else notes there are really only two actual language differences:

- `struct` defaults to public access and `class` defaults to private access.
- When inheriting, `struct` defaults to `public` inheritance and `class` defaults to `private` inheritance. (Ironically, as with so many things in C++, the default is backwards: `public` inheritance is by far the more common choice, but people rarely declare `struct`s just to save on typing the "`public`" keyword.

But the real difference in practice is between a `class`/`struct` that declares a constructor/destructor and one that doesn't. There are certain guarantees to a "plain-old-data" POD type, that no longer apply once you take over the class's construction. To keep this distinction clear, many people deliberately only use `struct`s for POD types, and, if they are going to add any methods at all, use `class`es. The difference between the two fragments below is otherwise meaningless:

```
class X
{
  public:

  // ...
};

struct X
{
  // ...
};
```

(Incidentally, here's a thread with some good explanations about what "POD type" actually means: What are POD types in C++?)

edited May 23 '17 at 11:47 Community ♦ 1 ●1

answered Jul 28 '09 at 4:02 quark 11.7k ●3 ●31 ●28

Nice example regarding the inheritance differences: here. – Liran Orevi Sep 13 '10 at 13:33

3 Whether you use `struct` or `class` has no bearing on whether your object is POD or whether you should define a copy constructor / destructor. Member functions also have no bearing on something being POD. As I re-

@DavidStone Basically, POD structs are supposed to be guaranteed to be backwards-compatible with C code, and thus are basically supposed to just be designed as C-style structs. – Justin Time Feb 21 '16 at 0:16

The "real difference" part of this answer is totally wrong. – juanchopanza Jul 29 '17 at 23:31

---

There are lots of misconceptions in the existing answers.

**Both `class` and `struct` declare a class.**

Yes, you may have to rearrange your access modifying keywords inside the class definition, depending on which keyword you used to declare the class.

But, beyond syntax, the *only* reason to choose one over the other is convention/style/preference.

Some people like to stick with the `struct` keyword for classes without member functions, because the resulting definition "looks like" a simple structure from C.

Similarly, some people like to use the `class` keyword for classes with member functions and `private` data, because it says "class" on it and therefore looks like examples from their favourite book on object-oriented programming.

The reality is that this completely up to you and your team, and it'll make literally no difference whatsoever to your program.

The following two classes are absolutely equivalent in every way except their name:

```
struct Foo
{
   int x;
};

class Bar
{
public:
   int x;
};
```

You can even switch keywords when redeclaring:

```
class Foo;
struct Bar;
```

(although some compilers will emit a warning when you do this, on the assumption that you probably didn't *intend* to do something so confusing and that you should therefore be prompted to double-check your code.)

and the following expressions both evaluate to true:

```
std::is_class<Foo>::value
std::is_class<Bar>::value
```

Do note, though, that you can't switch the keywords when *redefining*; this is only because (per the one-definition rule) duplicate class definitions across translation units must *"consist of the same sequence of tokens"*. This means you can't even exchange `const int member;` with `int const member;`, and has nothing to do with the semantics of `class` or `struct`.

edited Feb 15 at 12:39

answered Apr 28 '16 at 14:17
Lightness Races in Orbit
260k ●46 ●411 ●703

5 This was very informative, and honestly my favorite answer. Everyone else is treating them as separate entities, when underneath the hood, they are the same. I wonder if a struct definition in the Arduino environment is considered a cpp class, seeing as it's compiled using g++. – Ben Apr 24 '17 at 5:07

2 @Ben Of course it is. The Arduino compiler compiles C++; that's the end of it. – underscore_d May 22 '17 at 16:52

So if I'm understanding this right: As long as your visibility modifiers are explicit, not omitted, it doesn't matter. You could write a huge C++ application using nothing but structs; or you could go through and change every case of struct to class. As long as you're not using default visibility, the applications would be precisely the same. – Ryan Lundy Sep 29 '17 at 16:36

I'm not convinced that one translation unit of a C++ program can have the complete declaration `class foo { public: ... };` and another can have `struct foo { ... };` which would have to hold true according to the "absolutely equivalent" claim. It comes to reason that the incomplete declarations `struct foo;` and `class foo;` are interchangeable. These do not specify the class body, and so they speak nothing to the access layout. – Kaz Feb 15 at 2:36

@Kaz: You're right - if the definitions were for literally the same type, they must be lexically identical for the behaviour to be well-defined. The *struct-key* and *class-key* are otherwise logically exchangeable though (where the semantics are not affected), and `Foo` and `Bar` are still equivalent/identical types. I did make sure to say "when *redeclaring*" and give an example. Come to think of it, I'll clarify this in the answer to make sure I'm not misleading people into UB – Lightness Races in Orbit Feb 15 at 12:31

---

The only time I use a struct instead of a class is when declaring a functor right before using it in a function call and want to minimize syntax for the sake of clarity. e.g.:

```
struct Compare { bool operator() { ... } };
std::sort(collection.begin(), collection.end(), Compare());
```

edited Sep 10 '08 at 19:18

answered Sep 10 '08 at 18:08
Ferruccio
75.6k ●34 ●191 ●276

25 Now that it is several years later and C++11 is supported by all the major compilers, Lambdas make this even more concise. – user439793 Aug 14 '14 at 1:07

---

From the C++ FAQ Lite:

> The members and base classes of a struct are public by default, while in class, they default to private. Note: you should make your base classes explicitly public, private, or protected, rather than relying on the defaults.
>
> struct and class are otherwise functionally equivalent.
>
> OK, enough of that squeaky clean techno talk. Emotionally, most developers make a strong distinction between a class and a struct. A struct simply feels like an open pile of bits with very little

in the way of encapsulation or functionality. A class feels like a living and responsible member of society with intelligent services, a strong encapsulation barrier, and a well defined interface. Since that's the connotation most people already have, you should probably use the struct keyword if you have a class that has very few methods and has public data (such things do exist in well designed systems!), but otherwise you should probably use the class keyword.

edited Sep 10 '15 at 9:19
Sumurai8
11.3k ● 7 ● 28 ● 57

answered Jul 28 '09 at 4:07
Tal Pressman
5,552 ● 23 ● 32

I don't understand why they are stating that struct and class are functionally the same, but saying to prefer the one over the other in certain cases without any reasoning.. – deetz Oct 12 '17 at 6:33

1 The reason is convention. The compiler doesn't care which one you use, but another developer looking at your code will have an easier time understanding what you meant. – Tal Pressman Oct 13 '17 at 7:43

@deetz: The entire third paragraph is reasoning. – Lightness Races in Orbit Feb 26 at 2:13

---

One place where a struct has been helpful for me is when I have a system that's receiving fixed format messages (over say, a serial port) from another system. You can cast the stream of bytes into a struct that defines your fields, and then easily access the fields.

```
typedef struct
{
    int messageId;
    int messageCounter;
    int messageData;
} tMessageType;

void processMessage(unsigned char *rawMessage)
{
    tMessageType *messageFields = (tMessageType *)rawMessage;
    printf("MessageId is %d\n", messageFields->messageId);
}
```

Obviously, this is the same thing you would do in C, but I find that the overhead of having to decode the message into a class is usually not worth it.

edited Dec 12 '16 at 11:15
start2learn
23 ● 2 ● 7

answered Sep 10 '08 at 16:43
mbyrne215
1,166 ● 4 ● 11 ● 16

The same can be achieved in C. – Eugene Bujak Jun 16 '09 at 8:48

8 Or you could just implement `operator >>` on a class instead of writing the `processMessage` function, which would make your C++ look more like proper C++ and less like C. – Nick Bastin Jun 20 '10 at 18:29

1 Besides the fact that it's not portable between different systems, this violates aliasing rules, so it's not guaranteed to work even within a single architecture. – underscore_d May 22 '17 at 16:58

@underscore_d This can work platform (but not compiler) independent, but you must used fixed bitfields, a `__packed__` struct and have an endian-aware ifdef implementaion (you need to flip the order on a machine where the endianess is opposite what the external data source is providing). It's not pretty, but I have used to pack/unpack registers for remote peripherals on embedded platforms in a portable way. – crasic Dec 11 '17 at 19:45 ✎

@crasic But that's not really "portable", is it? After going to all the effort to lock yourself into one implementation like that, do you really feel it was quicker than simply writing standard-compliant code to do the unpacking? Serious question. I've gone to some lengths to write such code myself, and it ultimately wasn't too hard and was very educational. – underscore_d Dec 12 '17 at 9:44

---

You can use "struct" in C++ if you are writing a library whose internals are C++ but the API can be called by either C or C++ code. You simply make a single header that contains structs and global API functions that you expose to both C and C++ code as this:

```
// C access Header to a C++ library
#ifdef __cpp
extern "C" {
#endif

// Put your C struct's here
struct foo
{
    ...
};
// NOTE: the typedef is used because C does not automatically generate
// a typedef with the same name as a struct like C++.
typedef struct foo foo;

// Put your C API functions here
void bar(foo *fun);

#ifdef __cpp
}
#endif
```

Then you can write a function bar() in a C++ file using C++ code and make it callable from C and the two worlds can share data through the declared struct's. There are other caveats of course when mixing C and C++ but this is a simplified example.

edited Sep 20 '11 at 21:53

answered Oct 27 '09 at 1:50
Adisak
5,015 ● 28 ● 41

2 Best fitting answer. C-compatibility is really the most important reason. all other stuff like default access is esoteric. – Valentin Heinitz May 31 '15 at 22:32

---

As every one says, the only real difference is the default access. But I particularly use struct when I don't want any sort of encapsulation with a simple data class, even if I implement some helper methods. For instance, when I need something like this:

```
struct myvec {
    int x;
    int y;
    int z;

    int length() {return x+y+z;}
};
```

answered Jul 28 '09 at 16:13
ogoid
125 ● 1 ● 7

1 +1 for giving an example of a struct with some member functions which do not feel like you've "taken away the structness". – einpoklum Feb 5 '17 at 19:33

Structs (PODs, more generally) are handy when you're providing a C-compatible interface with a C++ implementation, since they're portable across language borders and linker formats.

If that's not a concern to you, then I suppose the use of the "struct" instead of "class" is a good communicator of intent (as @ZeroSignal said above). Structs also have more predictable copying semantics, so they're useful for data you intend to write to external media or send across the wire.

Structs are also handy for various metaprogramming tasks, like traits templates that just expose a bunch of dependent typedefs:

```
template <typename T> struct type_traits {
  typedef T type;
  typedef T::iterator_type iterator_type;
  ...
};
```

...But that's really just taking advantage of struct's default protection level being public...

edited Jan 31 '16 at 11:51
Deduplicator
31.2k ● 6 ● 45 ● 84

answered Sep 10 '08 at 17:46
argv0
646 ● 6 ● 11

1 That is not the correct usage of POD. A struct (or class) can be POD struct if (and only if) it contains ONLY POD members. – Martin York Mar 5 '09 at 8:07

4 "predictable copying semantics": The symantics are the same as for class (and have the same problems (shallow copy)). – Martin York Mar 5 '09 at 8:09

1 This post would have you believe (hopefully by accident) that all structs are PODs. This is not at all true. I hope that people are not mislead by this. – anthropomorphic Jul 30 '14 at 15:41

For C++, there really isn't much of a difference between structs and classes. The main functional difference is that members of a struct are public by default, while they are private by default in classes. Otherwise, as far as the language is concerned, they are equivalent.

That said, I tend to use structs in C++ like I do in C#, similar to what Brian has said. Structs are simple data containers, while classes are used for objects that need to act on the data in addition to just holding on to it.

answered Sep 10 '08 at 16:36
Andy
23.7k ● 4 ● 63 ● 78

To answer my own question (shamelessly), As already mentioned, access privileges are the only difference between them in C++.

I tend to use a struct for data-storage only. I'll allow it to get a few helper functions if it makes working with the data easier. However as soon as the data requires flow control (i.e. getters/setters that maintain or protect an internal state) or starts acquring any major functionality (basically more object-like), it will get 'upgraded' to a class to better communicate intent.

answered Sep 10 '08 at 16:50
Alan Hinchcliffe
3,533 ● 3 ● 13 ● 10

They are pretty much the same thing. Thanks to the magic of C++, a struct can hold functions, use inheritance, created using "new" and so on just like a class

The only functional difference is that a class begins with private access rights, while a struct begins with public. This is the maintain backwards compatibility with C.

In practice, I've always used structs as data holders and classes as objects.

answered Sep 10 '08 at 16:36
enigmatic
275 ● 1 ● 5

they're the same thing with different defaults (private by default for `class`, and public by default for `struct`), so in theory they're totally interchangeable.

so, if I just want to package some info to move around, I use a struct, even if i put a few methods there (but not many). If it's a mostly-opaque thing, where the main use would be via methods, and not directly to the data members, i use a full class.

answered Jul 28 '09 at 3:49
Javier
48.9k ● 7 ● 65 ● 112

Structs by default have public access and classes by default have private access.

Personally I use structs for Data Transfer Objects or as Value Objects. When used as such I declare all members as const to prevent modification by other code.

answered Jul 28 '09 at 3:51
anio
5,133 ● 4 ● 26 ● 45

When would you choose to use struct and when to use class in C++?

I use `struct` when I define `functors` and `POD`. Otherwise I use `class`.

```
// '()' is public by default!
struct mycompare : public std::binary_function<int, int, bool>
{
    bool operator()(int first, int second)
    { return first < second; }
};

class mycompare : public std::binary_function<int, int, bool>
{
public:
    bool operator()(int first, int second)
    { return first < second; }
};
```

answered Jul 28 '09 at 20:40
AraK
59.8k ● 27 ● 147 ● 221

This answer is showing signs of age :) `std::binary_function<>` is not just deprecated, c++17 even removes it. – sehe Nov 12 '17 at 0:23

Not a surprise really since it was written in the time of C++03, a language defined 15 years ago. – Lightness Races in Orbit Feb 26 at 2:15

**Class.**

Class members are private by default.

```
class test_one {
    int main_one();
};
```

Is equivalent to

```
class test_one {
  private:
    int main_one();
};
```

So if you try

```
int two = one.main_one();
```

We will get an error: `main_one is private` because its not accessible. We can solve it by initializing it by specifying its a public ie

```
class test_one {
  public:
    int main_one();
};
```

**Struct.**

A struct is a class where members are public by default.

```
struct test_one {
    int main_one;
};
```

Means `main_one` is private ie

```
class test_one {
  public:
    int main_one;
};
```

I use structs for data structures where the members can take any value, it's easier that way.

edited Nov 12 '17 at 0:09
sehe
254k ● 32 ● 306 ● 424

answered Nov 7 '17 at 22:41
Frrank
240 ● 1 ● 4 ● 19

I use struct only when I need to hold some data without any member functions associated to it (to operate on the member data) and to access the data variables directly.

Eg: Reading/Writing data from files and socket streams etc. Passing function arguments in a structure where the function arguments are too many and function syntax looks too lengthy.

Technically there is no big difference between class and struture except default accessibility. More over it depends on programming style how you use it.

answered Jul 28 '09 at 4:29
harik
186 ● 1 ● 3 ● 11

Technically both are the same in C++ - for instance it's possible for a struct to have overloaded operators etc.

However :

I use structs when I wish to pass information of multiple types simultaneously I use classes when the I'm dealing with a "functional" object.

Hope it helps.

```
#include <string>
#include <map>
using namespace std;

struct student
{
    int age;
    string name;
    map<string, int> grades
};

class ClassRoom
{
    typedef map<string, student> student_map;
  public :
    student getStudentByName(string name) const
    { student_map::const_iterator m_it = students.find(name); return m_it-
>second; }
  private :
    student_map students;
};
```

For instance, I'm returning a struct student in the get...() methods over here - enjoy.

answered Jul 28 '09 at 4:42
Maciek
8,175 ● 14 ● 50 ● 77

I use structs when I need to create POD type or functor.

answered Sep 15 '16 at 9:30
ivan.ukr
640 ● 4 ● 14

An advantage of `struct` over `class` is that it save one line of code, if adhering to "first public members, then private". In this light, I find the keyword `class` useless.

Here is another reason for using only `struct` and never `class`. Some code style guidelines for C++ suggest using small letters for function macros, the rationale being that when the macro is converted to an inline function, the name shouldn't need to be changed. Same here. You have your nice C-style struct and one day, you find out you need to add a constructor, or some convenience method. Do you change it to a `class`? Everywhere?

Distinguishing between `struct`s and `class`es is just too much hassle, getting into the way of doing what we should be doing - programming. Like so many of C++'s problems, it arises out of the strong desire for backwards compatability.

edited Dec 10 '17 at 9:49

answered Nov 14 '17 at 21:08
Vorac
2,787 ● 4 ● 27 ● 63

Why would you need to change it to a `class`? Did you think that a class defined with the `struct` keyword can't have member functions or a constructor? – Lightness Races in Orbit Feb 26 at 2:16

@LightnessRacesinOrbit because of 1. consistency and 2. some static analysers complain about the violation of 1. – Vorac Feb 27 at 15:27

That doesn't follow. What other "consistencies" do you adhere to? Every class with a member called `joe()` must be defined with `class` keyword? Every class with at least 4 `int` members must be defined with `struct` keyword? – Lightness Races in Orbit Feb 27 at 17:11

I thought that Structs was intended as a Data Structure (like a multi-data type array of information) and classes was inteded for Code Packaging (like collections of subroutines & functions)..

:(

answered Jul 28 '09 at 4:07
GaiusSensei
914 ● 4 ● 19 ● 38

We're looking for long answers that provide some explanation and context. Don't just give a one-line answer; explain why your answer is right, ideally with citations. Answers that don't include explanations may be removed.

I never use "struct" in C++.

I can't ever imagine a scenario where you would use a struct when you want private members, unless you're willfully trying to be confusing.

It seems that using structs is more of a syntactic indication of how the data will be used, but I'd rather just make a class and try to make that explicit in the name of the class, or through comments.

E.g.

```
class PublicInputData {
    //data members
 };
```

answered Sep 10 '08 at 18:02
Baltimark
3,944 ● 10 ● 31 ● 33

According to me, a "syntactic indication of how data will be used" is a perfectly good reason to use a struct, especially if the alternative is to use a comment or a name in the classname. – Viktor Sehr Jun 26 '09 at 11:41

Wouldn't declaring a `struct` already be pretty explicit that the members of the class will be, by default, public? – wrongusername Sep 15 '11 at 4:57

Code-sharing with C is a very good reason. – Lightness Races in Orbit Feb 26 at 2:17

Access functions aren't the only difference between structs and classes. A struct is better for POD (Plain Old Data) types, since it is easier to manage for the compiler and the programmer. For beginners who don't use object-oriented programming, using a struct is fine.

answered Feb 21 '14 at 18:58
Eitan Myron
65 ● 2 ● 11

1 Can you provide a reference for that " since it is easier to manage for the compiler"? – Vorac Dec 10 '17 at 9:44