

[Main Page](#)

[Site Index](#)

[Report an Issue](#)

[About / Contact](#)

[Support LearnCpp](#)

S E A R C H

Google Search

## 8.12 — Static member functions

B Y A L E X O N S E P T E M B E R 1 8 T H , 2 0 0 7 | L A S T

### Static member functions

In the previous lesson on [static member variables](#), you learned that static member variables are member variables that belong to the class rather than objects of the class. If the static member variables are public, we can access them directly using the class name and the scope resolution operator. But what if the static member variables are private? Consider the following example:

```
1  class Something
   {
   private:
       static int s_value;

   };

   int Something::s_value = 1; // initializer, this is okay even though s_value is private since it's a definition

   int main()
   {
       // how do we access Something::s_value since it is private?
   }
```



11  
12

13

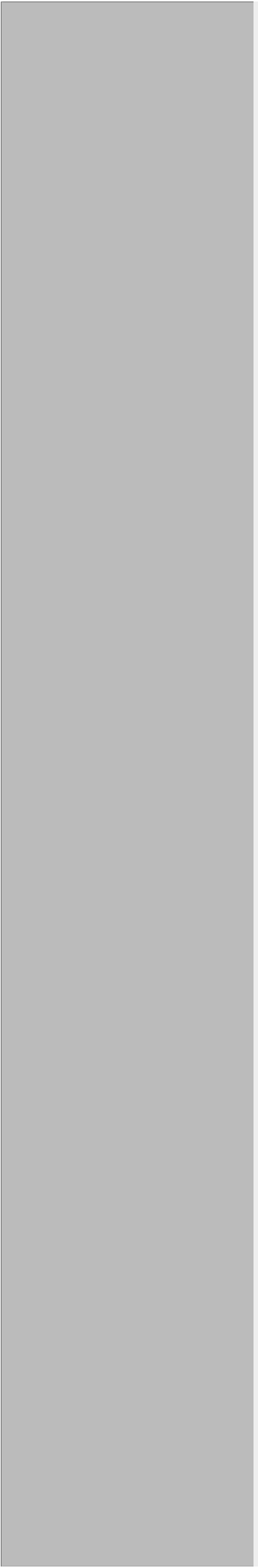
In this case, we can't access `Something::s_value` directly from `main()`, because it is private. Normally we access private members through public member functions. While we could create a normal public member function to access `s_value`, we'd then need to instantiate an object of the class type to use the function! We can do better. It turns out that we can also make functions static.

Like static member variables, static member functions are not attached to any particular object. Here is the above example with a static member function accessor:

```
1  class Something
   {
   private:
       static int s_value;
   public:
       static int getValue() { return s_value; } // static member function
   };

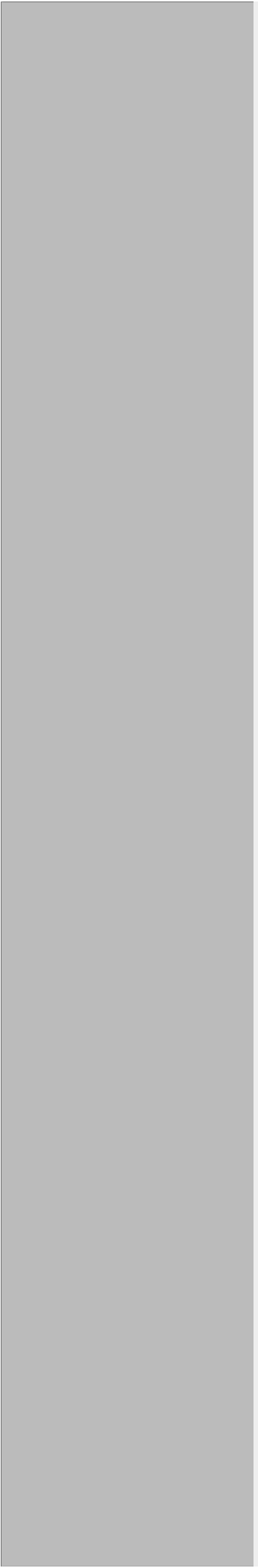
   int Something::s_value = 1; // initializer

   int main()
   {
       std::cout << Something::getValue() << '\n';
   }
```



2  
3  
  
4  
  
  
5  
  
6





7

8

9

10

11

12

13



Because static member functions are not attached to a particular object, they can be called directly by using the class name and the scope resolution operator. Like static member variables, they can also be called through objects of the class type, though this is not recommended.

### Static member functions have no *this* pointer

Static member functions have two interesting quirks worth noting. First, because static member functions are not attached to an object, they have no *this* pointer! This makes sense when you think about it -- the *this* pointer always points to the object that the member function is working on. Static member functions do not work on an object, so the *this* pointer is not needed.

Second, static member functions can directly access other static members (variables or functions), but not non-static members. This is because non-static members must belong to a class object, and static member functions have no class object to work with!

### Another example

Static member functions can also be defined outside of the class declaration. This works the same way as for normal member functions.

Here's an example:

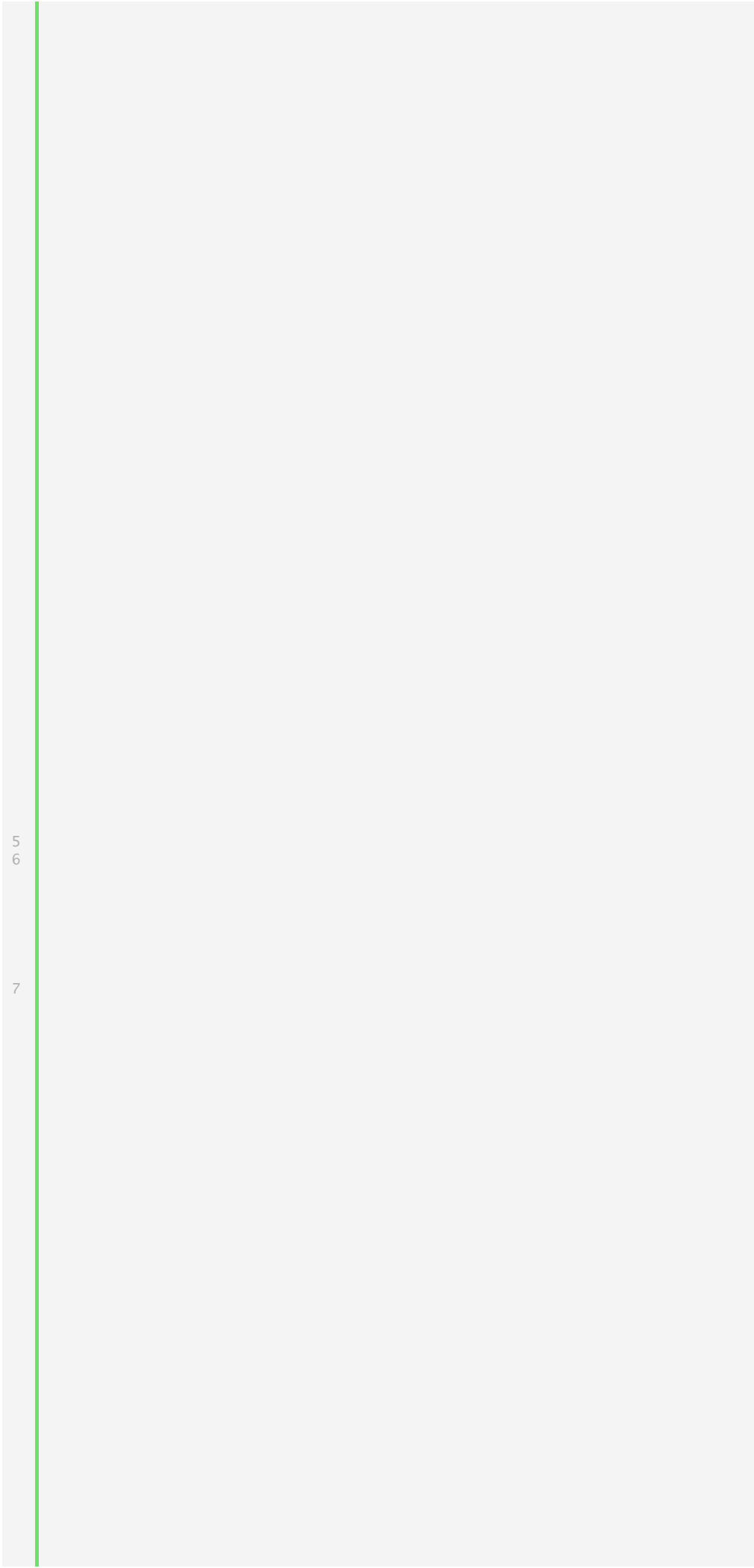
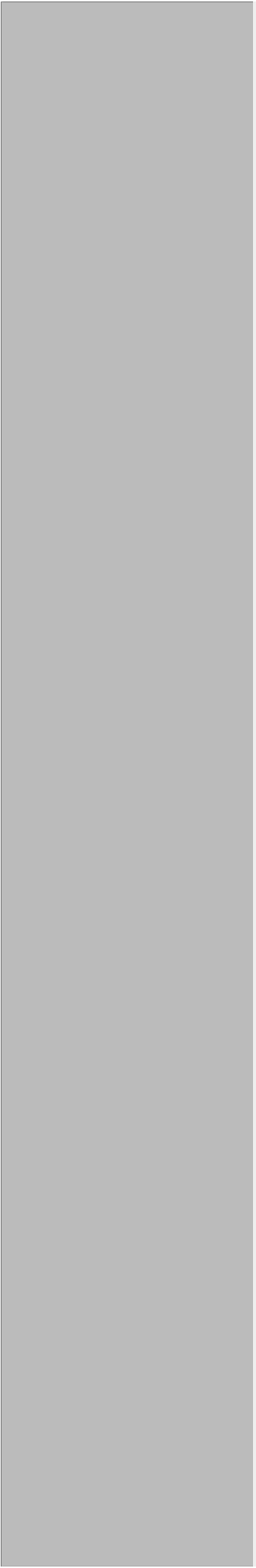
```
1  class IDGenerator
   {
   private:
       static int s_nextID; // Here's the declaration for a static member

   public:
       static int getNextID(); // Here's the declaration for a static function
   };

   // Here's the definition of the static member outside the class. Note we don't use
   // the static keyword here.
   // We'll start generating IDs at 1
   int IDGenerator::s_nextID = 1;

   // Here's the definition of the static function outside of the class. Note we don't
   // use the static keyword here.
2  int IDGenerator::getNextID() { return s_nextID++; }
3
   int main()
   {
       for (int count=0; count < 5; ++count)
           std::cout << "The next ID is: " << IDGenerator::getNextID() << '\n';

       return 0;
   }
4
```



8  
9  
10



11

12

13  
14

15

16  
17

18  
19



21  
22

23

This program prints:

The next ID is: 1  
The next ID is: 2  
The next ID is: 3  
The next ID is: 4  
The next ID is: 5

Note that because all the data and functions in this class are static, we don't need to instantiate an object of the class to make use of its functionality! This class utilizes a static member variable to hold the value of the next ID to be assigned, and provides a static member function to return that ID and increment it.

#### A word of warning about classes with all static members

Be careful when writing classes with all static members. Although such “pure static classes” (also called “monostates”) can be useful, they also come with some potential downsides.

First, because all static members are instantiated only once, there is no way to have multiple copies of a pure static class (without cloning the class and renaming it). For example, if you needed two independent IDGenerator objects, this would not be possible with a single pure static class.

Second, in the lesson on global variables, you learned that global variables are dangerous because any piece of code can change the value of the global variable and end up breaking another piece of seemingly unrelated code. The same holds true for pure static classes. Because all of the members belong to the class (instead of object of the class), and class declarations usually have global scope, a pure static class is essentially the equivalent of declaring functions and global variables in a globally accessible namespace, with all the requisite downsides that global variables have.

#### C++ does not support static constructors

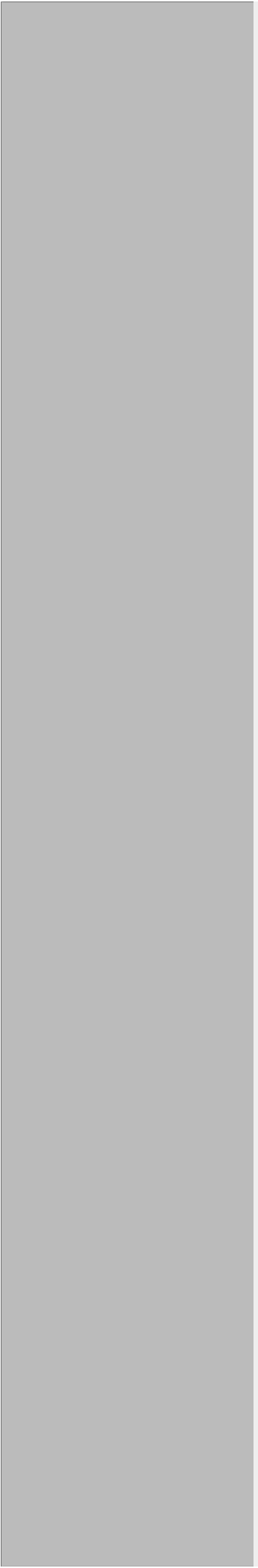
If you can initialize normal member variables via a constructor, then by extension it makes sense that you should be able to initialize static member variables via a static constructor. And while some modern languages do support static constructors for precisely this purpose, C++ is unfortunately not one of them.

If your static variable can be directly initialized, no constructor is needed: you can initialize the static member variable at the point of definition (even if it is private). We do this in the IDGenerator example above. Here's another example:

```
1  class MyClass
   {
   public:
       static std::vector<char> s_mychars;
   };

   std::vector<char> MyClass::s_mychars = { 'a', 'e', 'i', 'o', 'u' }; // initialize
                               static variable at point of definition
```

2  
3



4

5

6

7



If initializing your static member variable requires executing code (e.g. a loop), there are many different, somewhat obtuse ways of doing this. The following code presents one of the better methods. However, it is a little tricky, and you'll probably never need it, so feel free to skip the remainder of this section if you desire.

```
1 class MyClass
  {
  private:
    static std::vector<char> s_mychars;

  public:

    class _init // we're defining a nested class named _init
    {
    public:
      _init() // the _init constructor will initialize our static variable
      {
2       s_mychars.push_back('a');
3       s_mychars.push_back('e');
        s_mychars.push_back('i');
        s_mychars.push_back('o');
        s_mychars.push_back('u');
      }
    } ;

  private:
4   static _init s_initializer; // we'll use this static object to ensure the _init
    constructor is called
  };

```

```
std::vector<char> MyClass::s_mychars; // define our static member variable
MyClass::_init MyClass::s_initializer; // define our static initializer, which will c
all the _init constructor, which will initialize s_mychars
```

5  
6

7  
8



9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25



When static member `s_initializer` is defined, the `_init()` default constructor will be called (because `s_initializer` is of type `_init`). We can use this constructor to initialize any static member variables. The nice thing about this solution is that all of the initialization code is kept hidden inside the original class with the static member.

#### **Summary**

Static member functions can be used to work with static member variables in the class. An object of the class is

not required to call them.

Classes can be created with all static member variables and static functions. However, such classes are essentially the equivalent of declaring functions and global variables in a globally accessible namespace, and should generally be avoided unless you have a particularly good reason to use them.



[8.13 -- Friend functions and classes](#)



[Index](#)



[8.11 -- Static member variables](#)

ersky.fr  
remise sur la protection la plus  
plète. J'en profite.  
information >

.com  
marque de prêt-à-porter et  
cessoires 100% dédié à  
information >

rd-jeunesse.com  
onnement pour vos enfants  
information >

TRG AD

C + + T U P O R R I N A T L T H I S P O S T

## 96 comments to 8.12 — Static member functions

[« Older Comments](#) [1](#) [2](#)



sekhar

May 8, 2019 at 8:52 am · Reply

Hi Alex,

Why does static variable defined inside a static function(here \*inst of getInstance() function & val2 of printSample() static function) does not require it to be defined outside the class similar to any static variable declared as part of private/public section of a class(m\_id defined outside the class) ?

```
1  class Sample
   {
   private:
       static int m_id;
   public:

       Sample() { }

       ~Sample() { delete Sample::getInstance(); }

       static Sample* getInstance()
2  {
3      static Sample *inst = NULL;

       if(inst == NULL)
           inst = new Sample();

       return inst;
   }

4  static void printSample()
   {
       static int val2 = 50;
       std::cout << " Sample::printSample() " << m_id << " " << val2 << std::en
dl;
   }
};

//Sample* Sample::inst = NULL;
```

```
int Sample::m_id = 10;

int main()
{
    Sample *s = Sample::getInstance();

    Sample::printSample();

    return 0;
}
```

5

6  
7

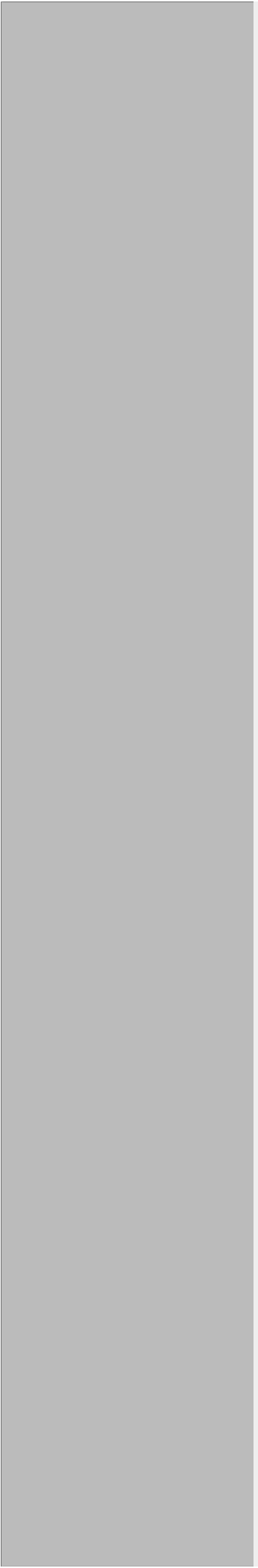
8

9

10

11





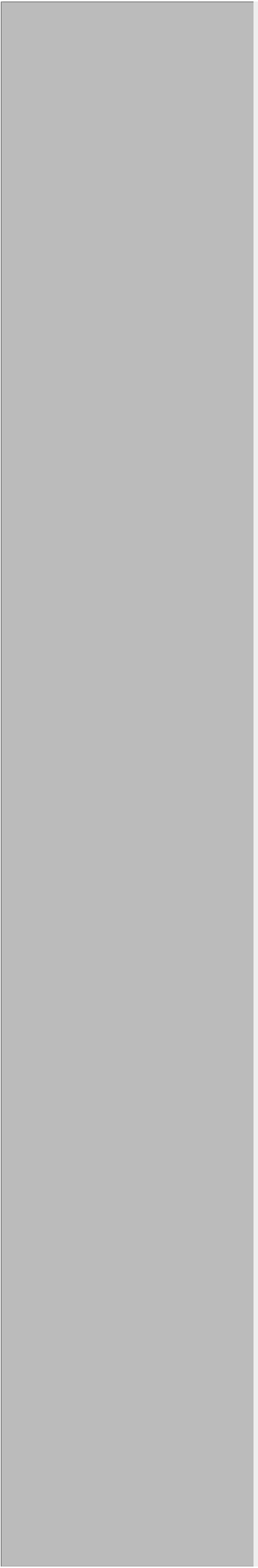
12

13

14

15





16

17

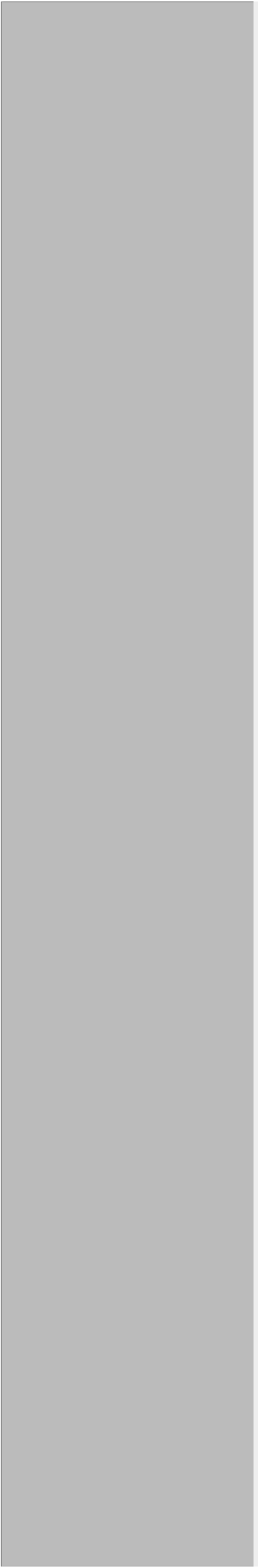
18

19

20

21



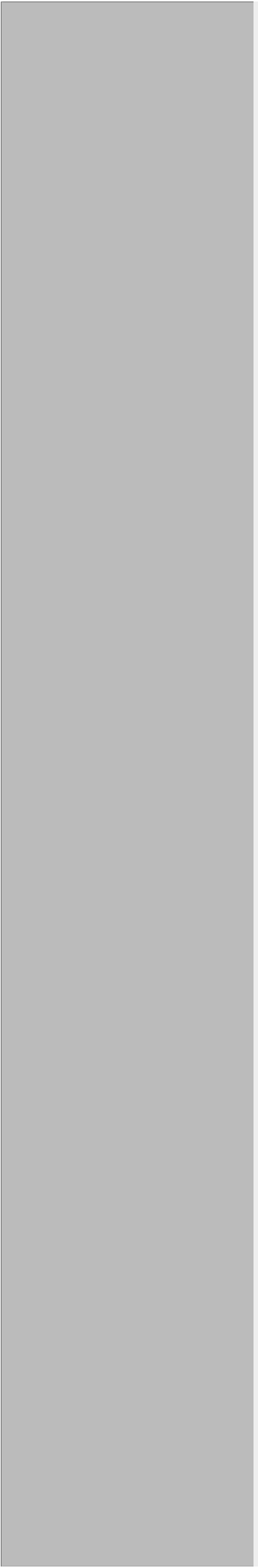


22

23

24





25

26

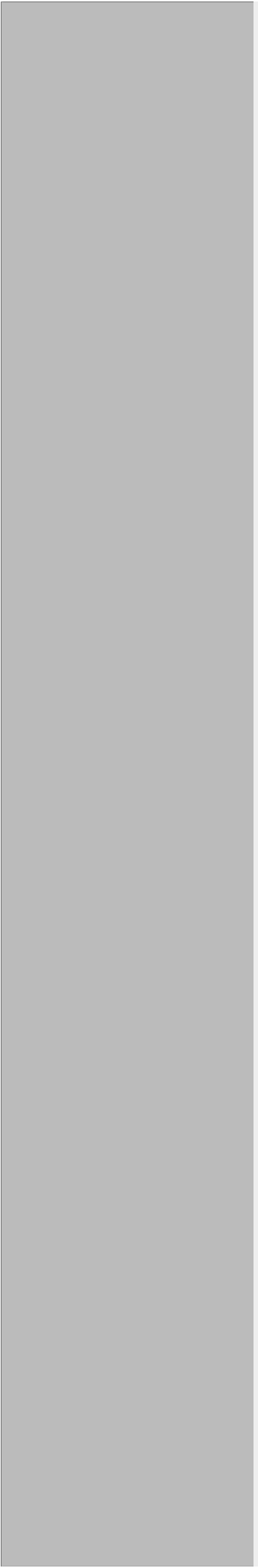
27

28

29

30





31

32

33

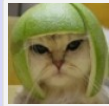
34

35

36

37



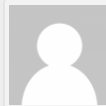


Alex

May 11, 2019 at 8:58 pm · Reply

In the above example, the "static int m\_id" inside class Sample serves as a declaration, not a definition (as no memory is allocated until the class is instantiated). Because static member objects can be accessed even without an object, the definition needs to be somewhere. Thus, it's defined outside the class.

In the case of inst, inst is a local static variable and is scoped to the function. Inst's declaration is also its definition, so no separate definition is needed.



Ishak

April 16, 2019 at 2:47 am · Reply

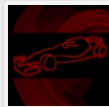
"First, because all static members are instantiated only once, there is no way to have multiple copies of a pure static class (without cloning the class and renaming it). For example, if you needed two independent IDGenerator objects, this would not be possible with a single pure static class."

Isn't this the case with normal classes too? I can't have two IDGenerator objects, since all the objects share the same static members.

"class declarations usually have global scope, a pure static class is essentially the equivalent of declaring functions and global variables in a globally accessible namespace, with all the requisite downsides that global variables have."

Yeah but is it really as bad as global variables, even though I have static member functions that dictate how we can interact with the static variables ?

thanks for the help



nascar driver

April 16, 2019 at 4:43 am · Reply

> Isn't this the case with normal classes too? I can't have two IDGenerator objects, since all the objects share the same static members.

This is the case for every class with static members.

> Yeah but is it really as bad as global variables

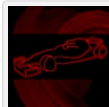
No, they're nested in a class and access can be controlled via functions. I think Alex exaggerated in this sentence.



tinu

February 18, 2019 at 3:45 pm · Reply

Are MyClass::s\_mychars and MyClass::s\_initializer being defined twice in the last code example?



nascar driver

February 19, 2019 at 8:26 am · Reply

Hi Tinu!

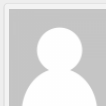
No, non-const static members cannot be initialized inside the class. Line 4 and 22 are just declarations. Line 25-26 define them, causing their constructors to be called.



tinu

February 19, 2019 at 12:23 pm · Reply

I see now, thanks a lot!



Gizmo

November 6, 2018 at 8:01 am · Reply

I'm trying to implement a static function member using class headers and .cpp files, and I'm having trouble getting it to work.

Here is a piece of example code demonstrating the issue.

Example.h:

```
1 | //imagine the proper header guards are included
```

```
class Example
{
public:

    static void example();
};
```

2  
3

4  
5

6  
7

Example.cpp:

```
1 //proper #includes and int main() are included.  
  static void Example::example()  
  {  
    std::cout << "This is the example.\n";  
  }
```

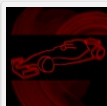


5

When I try to build the program, I get this error:

"error: cannot declare member function 'static void Example::example()' to have static linkage [-fpermissive]"

What am I doing wrong?



**nascardriver**

November 6, 2018 at 9:31 am · Reply

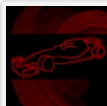
Remove the static keyword in the cpp file



**Nimbok**

May 6, 2018 at 3:16 pm · Reply

In the section on static constructors (the lack thereof), is there a reason to use an inner class instead of a static function to initialize the vector? An initializer function could be called multiple times, inappropriately, but you can do the same with the inner class approach by instantiating another one (MyClass::\_init ohno:). This would push another 'a', 'e', 'i', 'o', and 'u' onto the vector.



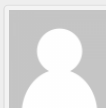
**nascardriver**

May 7, 2018 at 2:44 am · Reply

Hi Nimbok!

> is there a reason to use an inner class instead of a static function to initialize the vector?  
Using a class has the advantage that the constructor is automatically called when the object is created whereas we'd need to manually call a function.

> An initializer function could be called multiple times  
The @\_init class should be declared private, not public. If that was the case you wouldn't be able to create another @\_init object outside of @MyClass.  
Alternatively (but less efficient) one could check @s\_mychars.empty() to see if the vector has already been filled.



**Guozili**

February 2, 2018 at 5:01 am · Reply

Hi Alex,

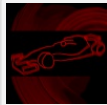
When you talk about warnings about classes with all static members, you mentioned that "First, because all static members are instantiated only once, there is no way to have multiple copies of a pure static class (without cloning the class and renaming it). For example, if you needed two independent IDGenerator objects, this would not be possible with a single pure static class."

But I still could not understand it, if I define:

```
1 IDGenerator A;
```

1 IDGenerator B;

Something wrong?



nascar driver

February 2, 2018 at 5:13 am · Reply

Hi Guozili!

The problem here is that A and B are equivalent. Take the following code for example

```
1  #include <iostream>

   class IDGenerator
   {
   private:
       static int s_nextID

   public:
       static int getNextID();
   };

   int IDGenerator::s_nextID = 1;
   int IDGenerator::getNextID() { return s_nextID++; }

   int main()
   {
       IDGenerator a{ };
       IDGenerator b{ };

2
3   std::cout << a.getNextID() << std::endl;
       std::cout << b.getNextID() << std::endl;

       return 0;
   }
```

4  
5

6

7  
8

9

10  
11  
12

13

14  
15

16  
17

18

19  
20

21

22  
23

24

#### Expected output

```
1 | 1
2 | 1
```

#### Actual output

```
1 | 1
2 | 2
```

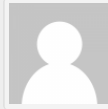
You cannot have two different IDGenerators with all static members.



Guozili

[February 2, 2018 at 7:08 am · Reply](#)

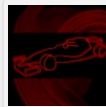
I suddenly get it, thank you!



david

[March 22, 2018 at 6:00 pm · Reply](#)

Hi nascardriver,  
doesn't that apply to classes that have both static and not-static members?



nascardriver

[March 23, 2018 at 2:01 am · Reply](#)

Can you give an example?

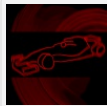
Whenever a class has a static member, this static member will be the same for all instances (Because it's not actually a member variable).



Ishak

[July 19, 2018 at 2:52 am · Reply](#)

I think what David is trying to ask, and what I want to ask also, is that does this problem apply to only all static classes, or is this issue present in a class containing both static and non-static members?

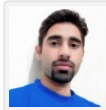


nascardriver

[July 19, 2018 at 3:04 am · Reply](#)

It's not a problem, it's a feature. It only turns into a problem when it's used in a wrong way.

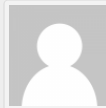
It only affects classes with static members, as those are members of the class itself and not of the objects.



Mr. Nobody

[November 6, 2017 at 8:04 pm · Reply](#)

Thanks alot...  
XXX



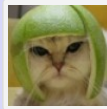
Luhan

[October 13, 2017 at 3:27 pm · Reply](#)

At the end of tutorial you show how to initialize a static vector through iteration, i didn't understand this:

```
1 | static _init s_initializer;
```

is this `_init` a class, and you create this member so when you define it at the same time the constructor is called and executes the push inside the vector?



Alex

October 20, 2017 at 12:05 pm · Reply

Yeah, `_init` is the nested class. The basic idea is to use the nested class constructor to initialize the outer class member.



Cunni

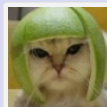
July 4, 2017 at 11:03 pm · Reply

Hi Alex,

In the paragraph of the A word of warning about classes with all static members section, you wrote:

"(...)Because all of the members belong to the class (instead of object of the class), and class declarations usually have global scope, a pure static class is essentially the equivalent of declaring functions and global variables in a globally accessible namespace, with all the requisite downsides that global variables have."

This only applies to static members that are within the public accessors, correct? If the static members are made private, than they are no longer in the global namespace?



Alex

July 5, 2017 at 3:41 pm · Reply

Well, the class members aren't in the global namespace either way (they're in the class namespace, which is in the global namespace) -- but the end result is similar, the members are accessible by everybody from everywhere.

If you make those members private, then they can no longer be accessed from outside the class, which helps alleviate the issue.

« Older Comments 1 2

### Leave a Comment

Put all code inside code tags: `[code]`your code here`[/code]`

**Name** (required)

**Email** (will not be published) (required)

**Website**

**Post Comment**

