8.11 — Static member variables

Posted by Alex

Review of static keyword uses

In the lesson on file scope and the static keyword, you learned that static variables keep their values and are not destroyed even after they go out of scope. For example:

```
#include <iostream>

int generateID()
{
    static int s_id = 0;
    return ++s_id;
}

int main()
{
    std::cout << generateID() << '\n';
    std::cout << generateID() << '\n';
    std::cout << generateID() << '\n';

    return 0;
}
```

This program prints:

```
1
2
3
```

Note that s_id has kept its value across multiple function calls.

The static keyword has another meaning when applied to global variables -- it gives them internal linkage (which restricts them from being seen/used outside of the file they are defined in). Because global variables are typically avoided, the static keyword is not often used in this capacity.

Static member variables

C++ introduces two more uses for the static keyword when applied to classes: static member variables, and static member functions. Fortunately, these uses are fairly straightforward. We'll talk about static member variables in this lesson, and static member functions in the next.

Before we go into the static keyword as applied to member variables, first consider the following class:

```
class Something
{
public:
    int m_value = 1;
};

int main()
{
    Something first;
    Something second;

    second.m_value = 2;

    std::cout << first.m_value << '\n';
    std::cout << second.m_value << '\n';

    return 0;
}
```

When we instantiate a class object, each object gets its own copy of all normal member variables. In this case, because we have declared two Something class objects, we end up with two copies of m_value: first.m_value, and second.m_value. first.m_nValue is distinct from second.m_nValue. Consequently, the program above prints:

```
1
2
```

Member variables of a class can be made static by using the static keyword. Unlike normal member variables, static member variables are shared by all objects of the class. Consider the following program, similar to the above:

```
class Something
{
public:
    static int s_value;
};

int Something::s_value = 1;

int main()
{
    Something first;
    Something second;

    second.s_value = 2;

    std::cout << first.s_value << '\n';
    std::cout << second.s_value << '\n';
    return 0;
}
```

This program produces the following output:

```
2
```

```
2
```

Because s_value is a static member variable, s_value is shared between all objects of the class. Consequently, first.s_value is the same variable as second.s_value. The above program shows that the value we set using first can be accessed using second!

Static members are not associated with class objects

Although you can access static members through objects of the class (as shown with first.s_value and second.s_value in the example above), it turns out that static members exist even if no objects of the class have been instantiated! Much like global variables, they are created when the program starts, and destroyed when the program ends.

Consequently, it is better to think of static members as belonging to the class itself, not to the objects of the class. Because s_value exists independently of any class objects, it can be accessed directly using the class name and the scope operator (in this case, Something::s_value):

```
1  class Something
2  {
3  public:
4      static int s_value; // declares the static member variable
5  };
6
7  int Something::s_value = 1; // defines the static member variable (we'll discuss t
8  his line below)
9
10 int main()
11 {
12     // note: we're not instantiating any objects of type Something
13
14     Something::s_value = 2;
15     std::cout << Something::s_value << '\n';
16     return 0;
   }
```

In the above snippet, s_value is referenced by class name rather than through an object. Note that we have not even instantiated an object of type Something, but we are still able to access and use Something::s_value. This is the preferred method for accessing static members.

Defining and initializing static member variables

When we declare a static member variable inside a class, we're simply telling the class that a static member variable exists (much like a forward declaration). Because static member variables are not part of the individual class objects (they get initialized when the program starts), you must explicitly define the static member outside of the class, in the global scope.

In the example above, we do so via this line:

```
1  int Something::s_value = 1; // defines the static member variable
```

This line serves two purposes: it instantiates the static member variable (just like a global variable), and optionally initializes it. In this case, we're providing the initialization value 1. If no initializer is provided, C++ initializes the value to 0.

Note that this static member definition is not subject to access controls: you can define and initialize the value even if it's declared as private (or protected) in the class.

If the class is defined in a .h file, the static member definition is usually placed in the associated code file for the class (e.g. Something.cpp). If the class is defined in a .cpp file, the static member definition is usually placed directly underneath the class. Do not put the static member definition in a header file (much like a global variable, if that header file gets included more than once, you'll end up with multiple definitions, which will cause a compile error).

There is one exception where a static member definition line is not required: when the static member is of type const integer or const enum. Those can be initialized directly on the line in which they are declared:

```
class Whatever
{
public:
    static const int s_value = 4; // a static const int can be declared and initial
ized directly
};
```

In the above example, because the static member variable is a const int, no explicit definition line is needed.

An example of static member variables

Why use static variables inside classes? One great example is to assign a unique ID to every instance of the class. Here's an example of that:

```
class Something
{
private:
    static int s_idGenerator;
    int m_id;

public:
    Something() { m_id = s_idGenerator++; } // grab the next value from the id gen
erator

    int getID() const { return m_id; }
};

// Note that we're defining and initializing s_idGenerator even though it is decla
red as private above.
// This is okay since the definition isn't subject to access controls.
int Something::s_idGenerator = 1; // start our ID generator with value 1

int main()
{
    Something first;
    Something second;
    Something third;

    std::cout << first.getID() << '\n';
    std::cout << second.getID() << '\n';
    std::cout << third.getID() << '\n';
    return 0;
}
```

This program prints:

```
1
2
3
```

Because s_idGenerator is shared by all Something objects, when a new Something object is created, the constructor grabs the current value out of s_idGenerator and then increments the value for the next object. This guarantees that each instantiated Something object receives a unique id (incremented in the order of creation). This can really help when debugging multiple items in an array, as it provides a way to tell multiple objects of the same class type apart!

Static member variables can also be useful when the class needs to utilize an internal lookup table (e.g. an array used to store a set of pre-calculated values). By making the lookup table static, only one copy exists for all objects, rather than making a copy for each object instantiated. This can save substantial amounts of memory.


8.12 -- Static member functions


Index


8.10 -- Const class objects and member functions

September 14, 2007

39 Replies

« Previous Next »

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Please enter an answer in digits:

3 × four =

Post Comment

Renu on January 15, 2008 at 10:29 am

static int s_nValue; ……
int Something::s_nValue = 1;

Why is “int” used in initialising
Something::s_nValue = 1

We have already declared s_nValue as static int in class definition. What is the purpose of using int again in
” int Something::s_nValue = 1; ” ?

Thanks,
Renu

Reply

Alex on January 15, 2008 at 10:43 am

This is a good question, and I am not sure of the answer. Because static members belong to the class, not objects of the class, I presume the declaration in the class is essentially treated as a forward declaration of the static member, which is actually defined in the body of the code. Thus, much like forward declarations for functions, which need to have the return type and parameter types in both the forward declaration and the actual definition, this would adhere to the same rules.

Reply

Dayu on March 18, 2009 at 12:24 pm

I tried the third code without the line

```
1   int Something::s_nValue=1;
```

. i.e.

```
1   #include <iostream>
2   class Something
3   {
4   public:
5       static int s_nValue;
6   };
7
8   //int Something::s_nValue;
9
10  int main()
11  {
12      Something cFirst;
13
14      Something cSecond;
15      std::cout << cSecond.s_nValue;
16
17      return 0;
18  }
19  <!--formatted-->
```

I got a compilation error:

…/main.cpp|15|undefined reference to `Something::s_nValue'|.

I also tried

```
1 int Something::s_nValue;
```

Note that there is no "=1" this time, and everything works out fine with s_nValue=0; Does this mean

```
1 int Something::s_nValue;
```

is always needed when we declare a static variable of a class? By the way, I am using Code::Blocks with GCC as the compiler on my Ubuntu.

Reply

Alex on May 1, 2009 at 7:42 pm

Static member variables always need to be defined in the code file. The fact that

```
1 int Something::s_nValue
```

happened to evaluate to 0 was probably circumstantial in the same way that

```
1 int nX;
```

will probably be 0, but it's not guaranteed.

Reply

Topjob on August 21, 2009 at 4:55 pm

Alex, then what did you mean by:

"In the absense of an initializing line, C++ will initialize the value to 0."?

Reply

Alex on January 21, 2016 at 8:36 pm

If you declare a static member variable but do not initialize it, C++ will initialize the static member variable to 0.

dog44wgm on November 15, 2011 at 2:45 pm

Definitely 0 according to the Standard (draft 2011):

http://www.open-std.org/Jtc1/sc22/wg21/docs/papers/2011/n3242.pdf

"Variables with static storage duration (3.7.1) or thread storage duration (3.7.2) shall be zero-initialized (8.5)
before any other initialization takes place".

Reply

Zafer on January 29, 2008 at 12:45 pm

Two questions. In the last coding example, why didn't we declare the objects cFirst, cSecond and cThird as const objects since we are using the GetId function, which has const in its declaration? My second question is about the static keyword. In the explanation provided after the last example, it says: "This guarantees that each Something object receives a unique id (incremented in the order of creation)." However since there is only one copy of the static variable and as we increment it, the id values of cFirst, cSecond and cThird will all be the same instead of being unique. This part needs to be clarified.

Reply

Alex on January 29, 2008 at 1:07 pm

cFirst, cSecond, and cThird could be declared as const objects since they only call GetID(), which is const.

The basic idea here is that there is only one copy of s_nIDGenerator that is shared amongst all of the objects of type Something. However, each individual object has it's own m_nID value that is not shared.

When we construct a new Something, the Something constructor copies the current value of s_nIDGenerator into the not-shared m_nID and then increments it. Keep in mind that when we increment s_nIDGenerator, this does not affect any of the m_nID values that have already been assigned!

So let's look at what happens in more detail. First, s_nIDGenerator is set to 1.

At this point we have:
s_nIDGenerator = 1

Then, we construct cFirst. cFirst's constructor copies the value of s_nIDGenerator (1) into cFirst.m_nID and then increments s_nIDGenerator to 2.

At this point we have:
cFirst.m_nID = 1
s_nIDGenerator = 2

Then, we construct cSecond. cSecond's constructor copies the value of s_nIDGenerator (2) into cSecond.m_nID, and then increments s_nIDGenerator to 3.

At this point we have:
cFirst.m_nID = 1
cSecond.m_nID = 2
s_nIDGenerator = 3

Finally, we construct cThird. This proceeds in the same way as the previous constructors.

At this point we have:
cFirst.m_nID = 1
cSecond.m_nID = 2
cThird.m_nID = 3
s_nIDGenerator = 4

Note that s_nIDGenerator is not used any more. It is only used to set the initial value of m_nID. If we ask a Something object for it's ID, it always returns it's non-shared m_nID value.

Reply

davidv on September 13, 2008 at 10:14 am

I have 2 questions.
First: in the last example, s_nIDGenerator was declared as a private member, but was accessed outside the class. I compiled the code, and it works. How come we didn't run into trouble?
Second: the name convention for static member does not reflect membership. Wouldn't something like ms_nName be better than s_nName?
Thank you.

Reply

luke on September 16, 2008 at 9:12 am

First: This is from Schildt's "The Complete Reference: C++ 4th ed."
"When you declare a **static** data me3mber within a class, you are not defining it. (That is, you are not allocating storage for it.) Instead, you must provide a global definition for it elsewhere, outside the class."
You're not accessing a private variable, but rather defining it. This also sheds some (more) light on Renu's question.

Second: Are you serious? I'm pretty sure Alex put the 's' and 'm' prefixes there to distinguish between static and member. 's' is definitely easier to distinguish from 'm' than "ms". Let's not forget these are examples. I'm surprised you didn't comment on "Something" not being a very good class name.

Reply

davidv on September 17, 2008 at 3:24 pm

Thanks a lot for your answers (I don't understand the sarcasm though, I wasn't trying to be picky).

Reply

Alex on September 21, 2008 at 9:37 am

Luke is correct on both accounts. I use s_ instead of ms_ because using ms_ would make it hard to distinguish ms_ from m_ variables.

Reply

luke on September 29, 2008 at 9:49 am

Sorry David. I don't think I got enough sleep that night. Hope there's no hard feelings.

Reply

Dawid Chemloul on August 6, 2010 at 5:09 am

I have questions and maybe some answers.

1. Is static in function variable guaranteed to be initialized (created) at first function call or is it compiler dependent?

as for singletons - it is possible to use static member to create singleton with nice encapsulation but I would rather use static variable in function if answer for my first question is yes.

Since if you will try make some static singleton objects - or some static objects using your singletons then initialisation order of all those global static globals need to be maintained with care.

Since it is possible that you will create a singleton instance before global member was initialised - then initialisation of global member will override previous usage… this seamnot to pose problems used this way in VS2010

Example:

```
1 class Singleton
2 {
3 public:
```

```
  static Singleton & getInstance()
  {
     static Singleton *instance = 0;
     if( ! instance )
     {
        instance = new Singleton();
     }
     return *instance;
  }
};
<!--formatted-->
```

Reply

Alex on January 21, 2016 at 9:15 pm

Local static variables are initialized the first time the definition line is encountered. Global static variables and static member functions are initialized at program startup.

Reply

Poke on August 27, 2010 at 9:46 pm

Hello! I'm having an issue with a static member variable…my g++ compiler is giving me the error: undefined reference to `ShipType::m_format'. Nearly all of the comments I've read say to instantiate the static outside the class, which I have done. Here's my code snippet:

```
//ShipType.h
class ShipType
{
public:
    enum Type {null, DD, SS, CA, BB, CV};
    enum Format {abbreviated, verbose};
...
    static string toString(ShipType::Type);
...
protected:
    static ShipType::Format m_format;
};

//ShipType.cpp
#include "ShipType.h"
...
ShipType::Format m_format = ShipType::verbose; //Here's where I've instantia
ted the static

string ShipType::toString(ShipType::Type t)
{
...
  if (ShipType::abbreviated == ShipType::m_format) //error message applies t
o this LOC
    retVal = shipStringsA[t];
  else
    retVal = shipStringsV[t];
  return retVal;
```

```
)//toString
```

Thank you in advance…

Poke

Reply

D.M. Ryan on September 8, 2010 at 2:51 am

Did you try changing

```
1 static ShipType::Format m_format;
```

to

```
1 static Format m_format;
```

?

In the examples above, the class scope isn't added when a static variable is declared within the class itself.

That fix may not work, but it's worth a try.

```
1 ShipType::Format m_format = ShipType::verbose;
```

is correct.

Reply

Alex on January 21, 2016 at 9:16 pm

```
1 ShipType::Format ShipType::m_format = ShipType::verbose;
```

Type, variable name (with class and scope resolution operator), and initializer.

Reply

Ashish79 on June 6, 2011 at 4:30 am

Hi,
Can we call non-static functions with static objects?

Reply

Alex on January 21, 2016 at 9:18 pm

Yes.

Reply

ashly on August 2, 2012 at 6:08 pm

like static member variables , static member functions , is static class itself possible in c++?

Reply

Alex on January 21, 2016 at 9:20 pm

No, C++ does not currently directly support applying the static keyword to classes.

The nearest thing is a class with only static variables and static functions.

Reply

Chris_M on March 18, 2014 at 9:09 am

Alex,

In the last paragraph of this section, you mention an internal lookup table. Are there any examples on this site on how to implement an internal lookup table? Thank you.

Reply

Alex on January 21, 2016 at 9:20 pm

This could be as simple as an initialized array of values.

Reply

Dan on March 1, 2015 at 3:32 pm

Great website!
One question - does it matter where exactly in the file the static member variable is initialised? For example, will it be initialised before int main() even if it is defined afterwards in the code?

Reply

Alex on January 21, 2016 at 9:22 pm

It doesn't matter where it's defined, so long as it's defined once and only once. The class definition

serves as a forward declaration, so it can be used from anywhere that can see the class declaration, and the linker will resolve it.

Reply

Mohammad Erfan on August 17, 2015 at 11:46 am

class Test
{
private:
int x,y;
static Test def;
public:
Test(int x=-1;int y=-1)
{
if(x==-1)
x=def.x;
if(y==-1)
y=def.y;
}
static void setTest(int xx,int yy);
};

My question is how can I set def value of Test.

Reply

Alex on August 17, 2015 at 12:24 pm

Add the following line outside of the class:

```
1 Test Test::def(2, 2);
```

Your constructor is pretty messed up though. You'll need to fix it before this will work.

Reply

Michael on September 1, 2015 at 5:51 pm

Last example:

```
1 class Something
2 {
3 private:
4     static int s_nIDGenerator;
5     int m_nID;
6 public:
```

```
7       Something() { m_nID = s_nIDGenerator++; }
8
9       int GetID() const { return m_nID; }
10  };
```

why is there a ‘const’ here:

```
1  int GetID() const { return m_nID; }
```

???

Reply

Alex on September 1, 2015 at 6:12 pm

That const means GetD() is guaranteed not to modify the object it’s associated with.

Besides being a generally good programming practice, it also allows GetID() to be called with a const object of type Something.

Reply

Avneet on September 3, 2015 at 1:08 am

```
1   class Something
2   {
3   public:
4       static int s_nValue;
5   };
6
7   int Something::s_nValue = 1;
8
9   int main()
10  {
11      Something::s_nValue = 2;
12      std::cout << Something::s_nValue;
13      return 0;
14  }
```

I was wondering why the data type is required when initializing the static member variable. We already declared the variable as int in the class declaration. If I leave the data type that is int in this case, the compiler throws an error, why? We are telling the compiler from where to pick (using :: operator) and also what to pick (using the identifier). Shouldn’t the compiler know what’s the data type of that variable?

Reply

Alex on September 3, 2015 at 11:58 pm

I don’t know for sure. But two things come to mind:
1) All definitions in C++ require types, and this is no exception.

2) If it didn't have a type, it would be dependent on a declaration having been previously seen, which could cause ordering issues.

Reply

Avneet on September 3, 2015 at 1:17 am

One more question, why we can't initialize that member variable using the scope operator inside a function?

This is an error:

```
1 int main()
2 {
3    int Something::s_nvalue = 20;
4 }
```

Reply

Alex on September 3, 2015 at 11:59 pm

Because the static member variable gets initialized at program startup (before main() runs). That means it needs to live in the global namespace, not inside a function.

Reply

Xi Yin on November 18, 2015 at 10:55 am

I have the same question! Thanks for the reply.

Reply

Xi Yin on November 18, 2015 at 10:49 am

Hi Alex,

Thank you for the wonderful tutorial. I have been following it for a week and everything that I used to confuse about just makes more sense now.

I have a question regarding the last example.

Since the objects are all non const and the function GetID() is const. why is it legal to call a const function with a non-const object, but not the other way round?

I tried a few things:
- remove the const in the function definition, -> it still works

- add const to all objects -> it works
- do both the above -> it does not work.

Thanks!

Reply

Alex on November 18, 2015 at 9:33 pm

A non-const object is one that can be modified. A const object is one that can't be modified.

A non-const function may modify the object it's called on. A const function guarantees it won't modify the object called on.

Therefore, a const object can only use const functions, because non-const functions could potentially modify the object, which the constness of the object won't allow.

However, non-const objects don't care if they get modified, so they can use both const and non-const functions.

Reply

GeekPro on February 5, 2016 at 3:10 pm

What are the real time example/ usage of static variables and member functions?

Reply

Main Page
Featured Articles

Report an Issue
About / Contact
Support LearnCpp

Search

Google Search

Language Selector

English

View Full Site

Proudly powered by WordPress