4.3 — Static duration variables

BY ALEX ON JUNE 19TH, 2007 | LAST MODIFIED BY ALEX ON APRIL 29TH, 2018

The static keyword is one of the most confusing keywords in the C++ language (maybe with the exception of the keyword “class”). This is because it has different meanings depending on where it is used.

In section **4.2 -- Global variables**, you learned that when applied to a variable declared outside of a block, it defines a global variable with internal linkage, meaning the variable could only be used in the file in which it was defined.

The static keyword can also be applied to variables declared inside a block, where it has a different meaning entirely. In section **4.1a -- Local variables and local scope**, you learned that local variables have automatic duration, which means they are created when the block is entered, and destroyed when the block is exited.

Using the static keyword on local variables changes them from automatic duration to static duration (also called fixed duration). A **static duration** variable (also called a “static variable”) is one that retains its value even after the scope in which it has been created has been exited! Static duration variables are only created (and initialized) once, and then they are persisted throughout the life of the program.

The easiest way to show the difference between automatic and static duration variables is by example.

Automatic duration (default):

```
#include <iostream>

void incrementAndPrint()
{
    int value = 1; // automatic duration by default
    ++value;
    std::cout << value << '\n';
} // value is destroyed here

int main()
{
    incrementAndPrint();
    incrementAndPrint();
    incrementAndPrint();
}
```

Each time incrementAndPrint is called, a variable named value is created and assigned the value of 1. incrementAndPrint increments value to 2, and then prints the value of 2. When incrementAndPrint is finished running, the variable goes out of scope and is destroyed. Consequently, this program outputs:

```
2
2
2
```

Now consider the static scope version of this program. The only difference between this and the above program is that we’ve changed the local variable value from automatic to static duration by using the static keyword.

Static duration (using static keyword):

```
#include <iostream>

void incrementAndPrint()
{
    static int s_value = 1; // static duration via static keyword.  This line is only executed once.
    ++s_value;
    std::cout << s_value << '\n';
} // s_value is not destroyed here, but becomes inaccessible

int main()
```

```
{
    incrementAndPrint();
    incrementAndPrint();
    incrementAndPrint();
}
```

In this program, because s_value has been declared as static, s_value is only created and initialized (to 1) once. When it goes out of scope, it is not destroyed. Each time the function incrementAndPrint() is called, the value of s_value is whatever we left it at previously. Consequently, this program outputs:

2
3
4

Just like we use “g_” to prefix global variables, it’s common to use “s_” to prefix static (static duration) variables. Note that internal linkage global variables (also declared using the static keyword) get a “g_”, not a “s_”.

One of the most common uses for static duration local variables is for unique identifier generators. When dealing with a large number of similar objects within a program, it is often useful to assign each one a unique ID number so they can be identified. This is very easy to do with a static duration local variable:

```
int generateID()
{
    static int s_itemID = 0;
    return s_itemID++; // makes copy of s_itemID, increments the real s_itemID, then returns the value i
n the copy
}
```

The first time this function is called, it returns 0. The second time, it returns 1. Each time it is called, it returns a number one higher than the previous time it was called. You can assign these numbers as unique IDs for your objects. Because s_itemID is a local variable, it can not be “tampered with” by other functions.

Static variables offer some of the benefit of global variables (they don’t get destroyed until the end of the program) while limiting their visibility to block scope. This makes them much safer for use than global variables.

Quiz

1) What effect does using keyword “static” have on a global variable? What effect does it have on a local variable?

Quiz Answers

1) **Hide Solution**

When applied to a global variable, the static keyword defines the global variable as having internal linkage, meaning the variable cannot be exported to other files.

When applied to a local variable, the static keyword defines the local variable as having static duration, meaning the variable will only be created once, and will not be destroyed until the end of the program.

 **4.3a -- Scope, duration, and linkage summary**

 **Index**

 **4.2a -- Why global variables are evil**