COP3503

The static Keyword

Welcome!

Placeholder for the instructor’s welcome message. Video team, please insert the instructor’s video here.

static Is a Modifier for Variables and Functions

- We can use additional keywords to assign properties to variables.

- One of those is static, and goes in front of the declaration of a variable or function.

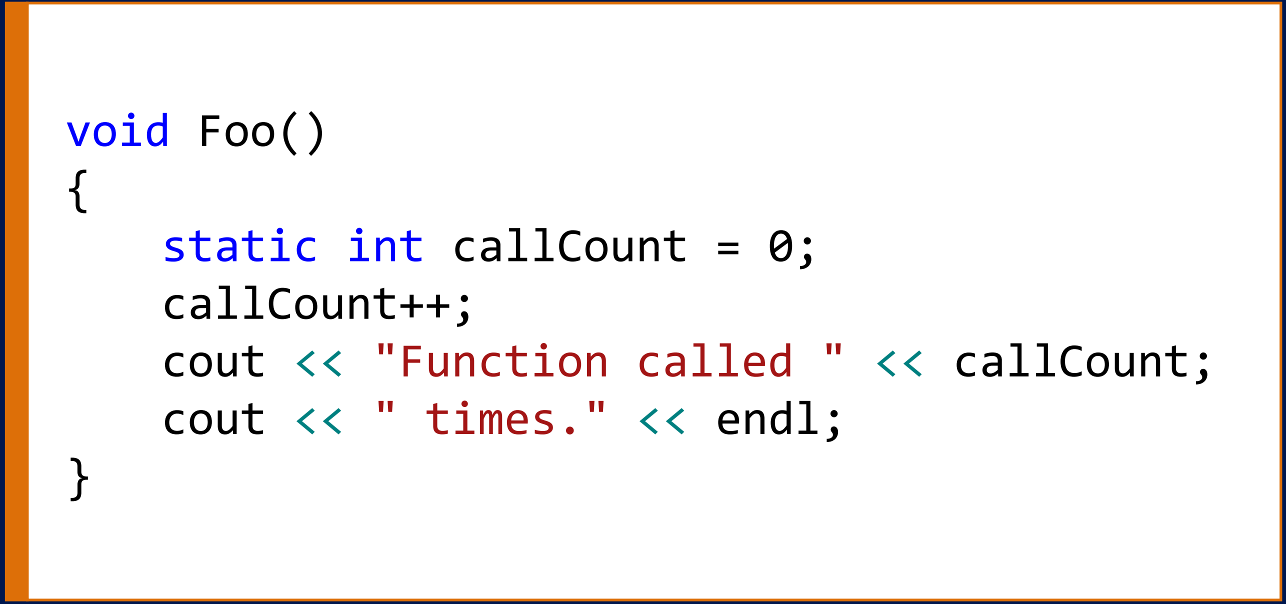
```
class Example
{
    static int number;
    static float price;
    int nonStaticNumber;
public:
    static void Foo();
    int Bar();
};
void Example::Foo()
{
    static int count = 0;
}
```

Question:
“So…what does this actually do?”

Answer:
Something slightly different in all 3 cases!

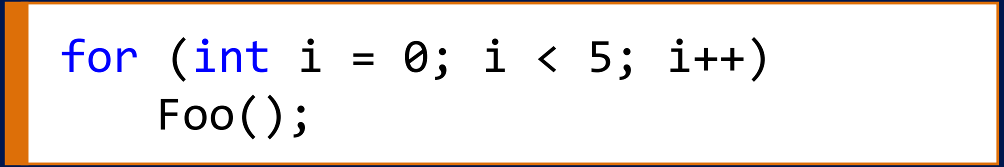
Static Local Variables

- Static variables are stored in memory differently than local variables.
- Static variables are initialized **once**, and then stay in memory for the rest of our program.

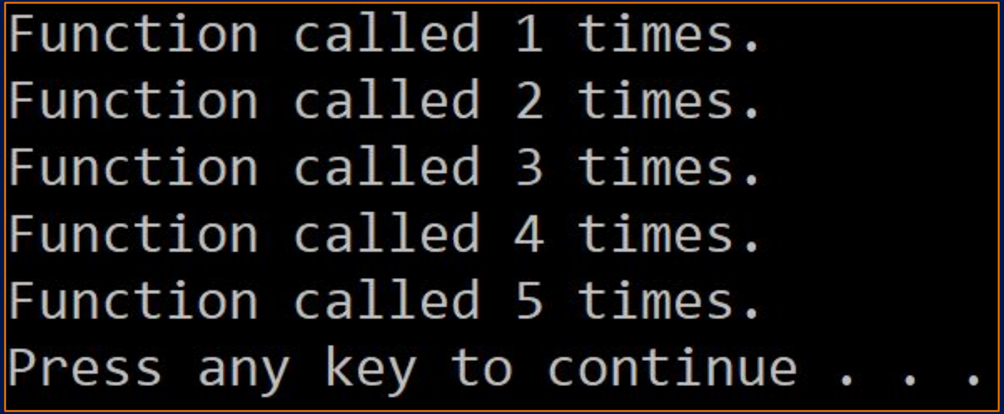


```cpp
void Foo()
{
    static int callCount = 0;
    callCount++;
    cout << "Function called " << callCount;
    cout << " times." << endl;
}
```

- Normal local variables will “fall out of scope” when the function ends.
- If the function is called again, the variable retains its value from previous function calls.



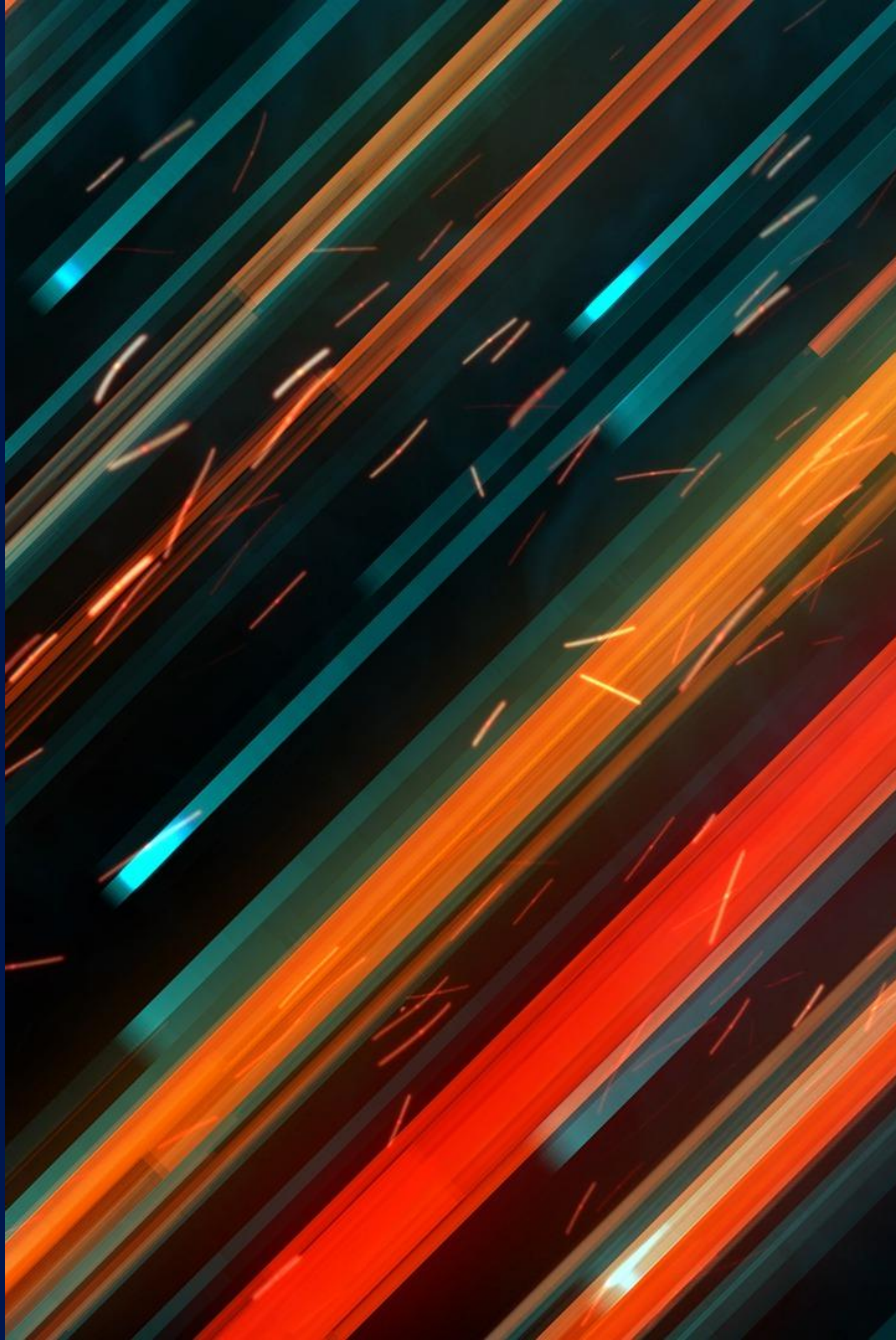
```cpp
for (int i = 0; i < 5; i++)
    Foo();
```



```
Function called 1 times.
Function called 2 times.
Function called 3 times.
Function called 4 times.
Function called 5 times.
Press any key to continue . . .
```

Static Member Variables

- Normally, each instance of a class has its own copy of member variables.
- A **static class member belongs to the class**, not individual instances.
- Only one copy of that variable exists.
- **All** instances of the class **share access to it**.

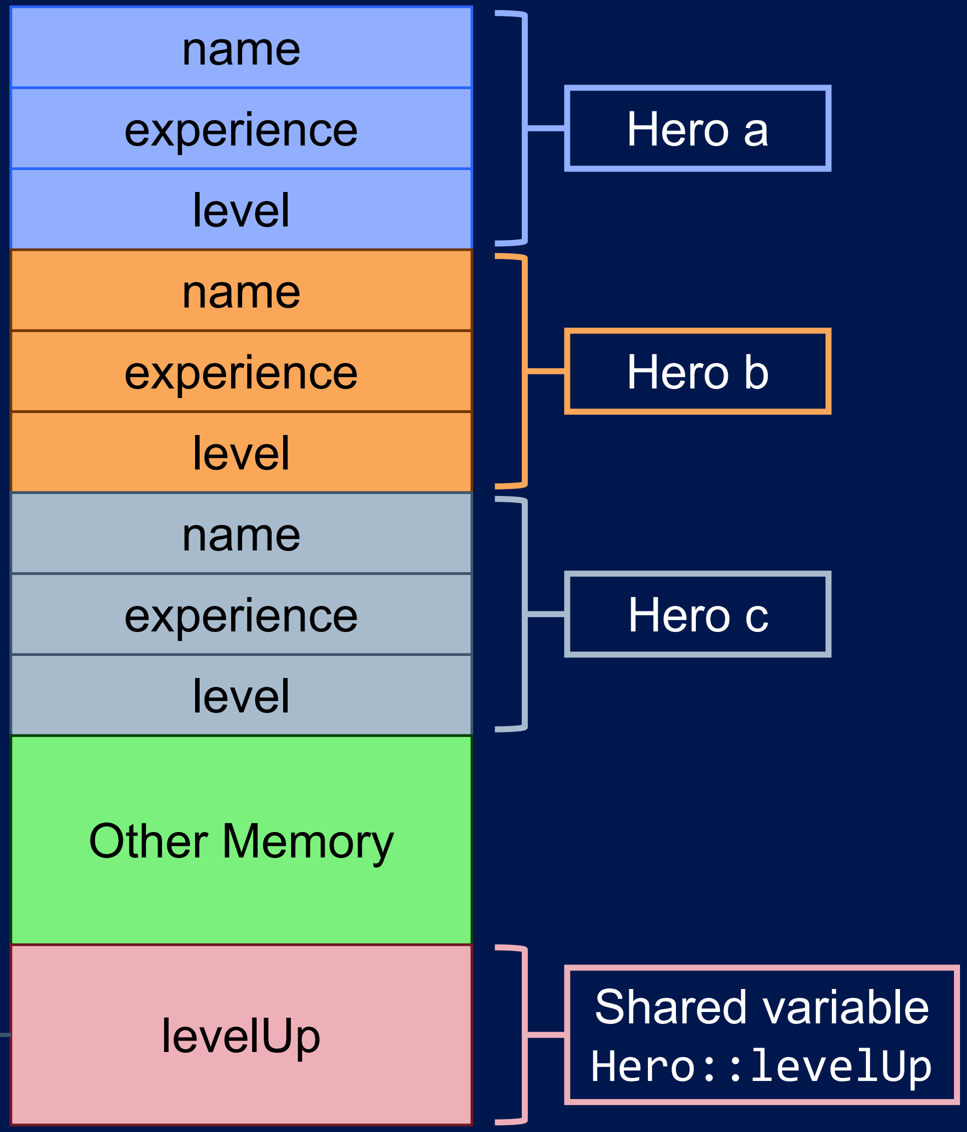


Static Member Variables

```
class Hero
{
    string name;
    int experience;
    int level;
public:
    // Experience points need to level up
    static int levelUp;
};

// Create 3 heroes
Hero a, b, c;

// Two ways to access the static variable
cout << a.levelUp << endl;
cout << Hero::levelUp << endl;
```



Initializing Static Class Variables

- Static variables must be redeclared and initialized outside of the class.

```
// File: Hero.h
class Hero
{
    /* omitted */
public:
    // Experience points need to level up
    static int levelUp;
};
```

```
// File: Hero.cpp
#include "Hero.h"


// Redeclare and initialize
int Hero::levelUp = 100;
```

```
class Hero
{
    /* omitted */
    // Alternate approach, must declare as const(ant)
    const static int levelUp = 100;
};
```

Static Member Functions

- Like static variables, static functions belong to the class, and not a specific instance of it.
- Static functions **aren’t invoked from an object**, but from the class.
- Static functions **have no “this” pointer**, and can’t access non-static member variables

```
class Example
{
public:
    static void Foo();
    void Bar();
};
```

```
Example object;
object.Bar();      // Invoke a member function

Example::Foo();    // Invoke a static member function
object.Foo();      // Technically this works too
```

Static Functions Can Only Access Static Class Members

- Static member functions don’t have a “`this`” pointer.
- Static member functions can’t access non-static member variables.

Static functions can only access static members. **Non-static functions** can access both.

```
class Example
{
    static int x;
    int y;
public:
    static void Foo();
    void Bar();
};
```

```
void Example::Foo()
{
    cout << x << endl;   // OK, static can access static
    cout << y << endl;   // Error, can't access non-static member
    Bar();               // Error, can’t access non-static member
}
```

```
void Example::Bar()
{
    cout << x << " " << y << endl;   // OK, can access static
    cout << Example::x << endl;      // Same thing
    Foo();                           // Or Example::Foo()
}
```

Why Use These?

- Sometimes we want to write a class for the usual reasons (encapsulation), but we don’t want or need more than one instance.
- It might be helpful to have “universal” access to that one instance of all the information.
- Imagine a utility class that prints debugging messages and writes them to a file.
- Access to that functionality outside of class instances would be useful.

Example: A Debug Log

```
class DebugLog
{
/* class variables, static or otherwise, here */
public:

    static void LogMessage(string msg); // Normal events
    static void LogError(string msg);   // Problematic events
    static void LogWarning(string msg); // Things that might become problematic...
};
```

No class object necessary to call these functions.

No instances of the class means no constructors—any initialization steps have to be handled manually.

Example: A Debug Log

```cpp
// AnyFileInYourCode.cpp
#include "DebugLog.h"

void SomeFunction()
{
    DebugLog::Message("Something just happened in the program");
}

void SomeOtherClass::MemberFunction()
{
    if (someCondition)
    {
        DebugLog::Error("Error! Something bad happened, probably a bug!");
        throw runtime_error("Critical error!");
    }
}
```

Recap

- **Static member variables** belong to a class.
  - Not to instances of the class, but the class itself
  - **One copy** exists in memory, any instances of the class share the variable.
  - This can help avoid creating copies
- **Static member functions** also belong to the class
  - They aren't invoked by an object, but by the class.
  - They **don't have a "this" pointer** and can't access non-static member variables.
  - A good way to get functionality without instantiating the class.
- They're just another programming tool at your disposal.

Conclusion

Placeholder for the instructor’s welcome message. Video team, please insert the instructor’s video here.

Thank you for watching.