4.3a — Scope, duration, and linkage summary

BY ALEX ON APRIL 19TH, 2016 | LAST MODIFIED BY ALEX ON JULY 17TH, 2017

The concepts of scope, duration, and linkage cause a lot of confusion, so we're going to take an extra lesson to summarize everything.

Scope summary

An identifier's **scope** determines where it is accessible. An identifier that is out of scope can not be accessed.

- Variables with **block scope / local scope** can only be accessed within the block in which they are declared (including nested blocks). This includes:
  - Local variables
  - Function parameters
  - Locally-defined types
- Variables and functions with **global scope / file scope** can be accessed anywhere in the file. This includes:
  - Global variables
  - Normal functions
  - Globally-defined types

Duration summary

A variable's **duration** determines when it is created and destroyed.

- Variables with **automatic duration** are created at the point of definition, and destroyed when the block they are part of is exited. This includes:
  - Normal local variables
- Variables with **static duration** are created when the program begins and destroyed when the program ends. This includes:
  - Global variables
  - Static local variables
- Variables with **dynamic duration** are created and destroyed by programmer request. This includes:
  - Dynamically allocated variables (we'll talk about these when we cover dynamic allocation in chapter 6)

Linkage summary

An identifier's **linkage** determines whether multiple instances of an identifier refer to the same identifier or not.

- Identifiers with **no linkage** mean the identifier only refers to itself. This includes:
  - Normal local variables
  - User-defined types, such as enums, typedefs, and classes declared inside a block (we'll cover these in later lessons).
- Identifiers with **internal linkage** can be accessed anywhere within the file it is declared. This includes:
  - Static global variables (initialized or uninitialized)
  - Const global variables
  - Static functions (we'll cover these in chapter 7)
- Identifiers with **external linkage** can be accessed anywhere within the file it is declared, or other files (via a forward declaration). This includes:
  - Normal functions
  - Non-const global variables (initialized or uninitialized)
  - Extern const global variables
  - User-defined types, such as enums, typedefs, and classes declared in the global scope (we'll cover these in later lessons).

Identifiers with external linkage will cause a duplicate definition linker error if the definitions are compiled into more than one .cpp file.

There are a few things worth explicitly noting here. First, functions are extern by default, and can be made internal by using the static keyword.

Second, astute readers may note that global types have external linkage, but their definitions don't cause a linker error when included in multiple files. This is because types, templates, and extern inline functions have an exemption that allows them to be defined in more than one file, so long as the definitions are identical. Otherwise, they wouldn't be of much use.

Variable scope, duration, and linkage summary

Because variables have scope, duration, and linkage, let's summarize in a chart:

Type	Example	Scope	Duration	Linkage	Notes
Local variable	int x;	Block scope	Automatic duration	No linkage	
Static local variable	static int s_x;	Block scope	Static duration	No linkage	
Dynamic variable	int *x = new int;	Block scope	Dynamic duration	No linkage	
Function parameter	void foo(int x)	Block scope	Automatic duration	No linkage	
External non-const global variable	int g_x;	File scope	Static duration	External linkage	Initialized or uninitialized
Internal non-const global variable	static int g_x;	File scope	Static duration	Internal linkage	Initialized or uninitialized
Internal const global variable	const int g_x(1);	File scope	Static duration	Internal linkage	Must be initialized
External const global variable	extern const int g_x(1);	File scope	Static duration	External linkage	Must be initialized

Forward declaration summary

You can use a forward declaration to access a function or variable in another file:

Type	Example	Notes
Function forward declaration	void foo(int x);	Prototype only, no function body
Non-const global variable forward declaration	extern int g_x;	Must be uninitialized
Const global variable forward declaration	extern const int g_x;	Must be uninitialized


4.3b -- Namespaces


Index


4.3 -- Static duration variables

Share this:

C++ TUTORIAL | PRINT THIS POST

50 comments to 4.3a — Scope, duration, and linkage summary