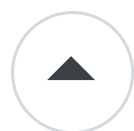


What is a variable's linkage and storage specifier?

Asked 16 years, 3 months ago Modified 16 years, 3 months ago

Viewed 6k times



When someone talks about a variables storage class specifier, what are they talking about?

11

They also often talk about variable linkage in the same context, what is that?



c++

c



Share

Improve this question

Follow

asked Sep 18, 2008 at 19:15



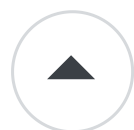
Benoit

38.9k ● 24 ● 85 ● 117

2 Answers

Sorted by:

Highest score (default)



33

The storage class specifier controls the *storage* and the *linkage* of your variables. These are two concepts that are different. C specifies the following specifiers for variables: auto, extern, register, static.



Storage



The storage duration determines how long your variable



will live in ram.



There are three types of storage duration: static, automatic and dynamic.

static

If your variable is declared at file scope, or with an `extern` or `static` specifier, it will have static storage. The variable will exist for as long as the program is executing. No execution time is spent to create these variables.

automatic

If the variable is declared in a function, but **without** the `extern` or `static` specifier, it has automatic storage. The variable will exist only while you are executing the function. Once you return, the variable no longer exist. Automatic storage is typically done on the stack. It is a very fast operation to create these variables (simply increment the stack pointer by the size).

dynamic

If you use `malloc` (or `new` in C++) you are using dynamic storage. This storage will exist until you call `free` (or `delete`). This is the most expensive way to create storage, as the system must manage allocation and deallocation dynamically.

Linkage

Linkage specifies who can see and reference the variable. There are three types of linkage: internal linkage, external linkage and no linkage.

no linkage

This variable is only visible where it was declared.
Typically applies to variables declared in a function.

internal linkage

This variable will be visible to all the functions within the file (called a [translation unit](#)), but other files will not know it exists.

external linkage

The variable will be visible to other translation units.
These are often thought of as "global variables".

Here is a table describing the storage and linkage characteristics based on the specifiers

Storage Class Specifier	Function Scope	File Scope

none linkage	automatic no linkage	static external
extern linkage	static external linkage	static external
static linkage	static no linkage	static internal
auto	automatic no linkage	invalid
register	automatic no linkage	invalid

Share Improve this answer

edited May 23, 2017 at 12:13

Follow



Community Bot

1 • 1

answered Sep 18, 2008 at 19:17



Benoit

38.9k • 24 • 85 • 117

-
- 1 Should perhaps include an extra bit of detail on translation units as they aren't just files... they are source files plus any headers included in them (however indirectly) – [workmad3](#) Sep 18, 2008 at 19:21
 - 1 Wow... +1. Note that "auto" will change its meaning with C++0x : en.wikipedia.org/wiki/C%2B%2B0x – [paercebal](#) Sep 18, 2008 at 19:49
 - 1 Update for c++11: thread storage duration. The object is allocated when the thread begins and deallocated when the thread ends. Each thread has its own instance of the object. Only objects declared `thread_local` have this storage duration. `thread_local` can appear together with `static` or `extern` to adjust linkage. – [Li Chen](#) Oct 31, 2017 at 13:25
-



0



Variable storage classes or type specifiers (like `volatile`, `auto` and `static`) define how/where variables are saved during program execution. For example, variables defined in functions are usually saved on the stack, which means that it will be lost after the function returns. Using the "static" keyword, you can force the compiler to put the variable in the data segment in memory, making the variables content persistent between calls to that function. The "register" keyword will cause the compiler to

try as hard as possible to put the variable in a CPU register, useful for counters in loops etc. However, it's not guaranteed that it's actually in a register after all.

Read more about type specifiers [here](#).

Share Improve this answer

answered Sep 18, 2008 at 19:26

Follow



[jkramer](#)

15.7k ● 5 ● 50 ● 48

For an odd definition of "try as hard as possible" which usually means "do nothing at all". `register` is completely obsolete except to document (by enforcement) that your code is not allowed to take the address of a variable.

– [R.. GitHub STOP HELPING ICE](#) Jan 22, 2011 at 20:13
