Princeton University
COS 217: Introduction to Programming Systems
C Variable Declarations and Definitions

A variable **declaration** is a statement that informs the compiler of the name, type, scope, linkage, and duration of the variable. A variable **definition** is a declaration that causes the compiler to allocate memory.

Scope (compile-time concept, a.k.a. Visibility)

File: The variable is accessible within the file in which it is declared, from the point of declaration to the end of the file.

Block: The variable is accessible within the block in which it is declared, from the point of declaration to the end of the block.

Two other scopes exist in the C standard that we will not see in COS 217: function and function prototype.

Linkage (link-time concept)

External: The variable is accessible from multiple files.

Internal: The variable is accessible from only the file in which it is declared. This is a simplification of the C standard, which differentiates "internal linkage" and "no linkage". See King 18.2 for a fuller discussion.

Storage Duration (run-time concept)

Automatic (a.k.a. Temporary): The variable exists in memory only during the execution of the function or block in which it is declared. The variable's value is stored in its activation record on the runtime Stack.

Static (a.k.a. Process): The variable exists in memory throughout the entire process. The variable's value is stored in the Data Section (if the programmer specifies a non-zero initial value) or the BSS Section (if the programmer does not specify an initial value)[†]. The variable's value is initialized at program startup. If in the BSS section, its initial value is 0.

C Code	Decl/Def	Scope	Linkage	Duration	Location
`int a = 5;`	definition	file	external	process	
`int b;`	definition[†]	file	external	process	
~~`extern int c = 5;`~~	definition	file	external	process	
`extern int d;`	declaration	file	external[†]	process	???
`static int e = 5;`	definition	file	internal	process	
`static int f;`	definition	file	internal	process	
`void fun(int g) {`	definition	block	internal	temporary	
`  int h = 5;`	definition	block	internal	temporary	
`  int i;`	definition	block	internal	temporary	
~~`  extern int j = 5;`~~	ILLEGAL				
~~`  extern int k;`~~	declaration	block	???	process	???
`  static int l = 5;`	definition	block	internal	process	
`  static int m;`	definition	block	internal	process	
`  ... }`					

[†] The location for variables with process duration that are initialized to zero in their definition is compiler dependent. Our current version of gcc217 stores them in the BSS section.
b: the C standard says this is a "tentative definition", which the compiler treats as the "real" definition so long as there is no duplicate declaration in this file. gcc217's linker treats this as a "real" definition. (See lines 10 and 11 on the second page of this handout.)
d: has the same linkage as a visible prior declaration; only *useful* for externally linked global variable from another file.

Examples of Global Variable Declarations and Definitions

Suppose a program consists of file1.c and file2.c (only). Consider these combinations of global variable declarations and definitions:

	file1.c	file2.c	Result
	Reasonable combinations:		
1	`static int i = 5;`	`static int i = 6;`	static def / static def => OK
2	`static int i = 5;`	`static int i;`	static def / static def => OK
3	`static int i;`	`static int i;`	static def / static def => OK
4	`int i = 5;`	`extern int i;`	def / decl => OK
5	`int i;`	`extern int i;`	def / decl => OK
	Less reasonable combinations:		
6	`int i = 5;`	`static int i = 6;`	def / static def => OK
7	`int i = 5;`	`static int i;`	def / static def => OK
8	`int i;`	`static int i = 6;`	def / static def => OK
9	`int i;`	`static int i;`	def / static def => OK
	Erroneous combinations:		
10	`int i = 5;`	`int i;`	def / tentative def => Undefined in C standard: error in gcc217, but some compilers and linkers will treat file2's i as a declaration (like line 4 above)
11	`int i;`	`int i;`	tentative def / tentative def => Undefined in C standard: error in gcc217, but some compilers and linkers will treat the first i seen as a definition and the other as a declaration (like line 5)
12	`int i = 5;`	`int i = 5;`	def / def => error
13	`extern int i;`	`extern int i;`	decl / decl => error
14	`extern int i;`	`static int i = 6;`	decl / static def => error
15	`extern int i;`	`static int i;`	decl / static def => error