4

Names

Exercises

4.1 Here's a summary:

Lang	Decl/Use	Overld	Array Size
Perl	*undef*	no	assign index
Python	assign	yes	asign/append
PHP	undef	no	assign index–size++
C	yes	no	static/malloc
C++	yes	yes	static/new
C#	yes	yes	new

Lang	Scopes	Where
Perl	def: global–my func–class	n/a
Python	LGB rule	before use
PHP	def: func–global decl	n/a
C	global/block	top of scope C99: before use
C++	+ class	before use
C#	C++	before use

4.2 a. By default any global name can be accessed in another compilation unit. Names are usually declared in a `.h` (or header) file.

b. By declaring them to be `static`. Note that globals are static. In order to avoid introducing another reserved word, the reserved word `static` is used, although it is potentially confusing.

c. To avoid name space clutter; to prevent another compilation unit from directly accessing or modifying global variables.

4.3 A definition refers to the place where a variable is created or assigned storage. A declaration refers to places where the nature of a variable is stated but no storage is allocated. Declarations commonly occur in header files, while definitions do not.

4.4 External declarations of variables, types and functions are commonly collected in a separate header file, which is included at the beginning of any source file that references these. Historically, this has occurred because compiled source files do not contain type information.

In contrast, Java stores type information in its `class` files, so that the need for a separate header file does not exist.

4.5 a. Types alone are insufficient. For example, if a and b are both declared `int` and `float`, then the computation in the statement `a += b` cannot be uniquely determined.

b. Consider the `print` method of `System.out`. It is overloaded for all the primitive types including `int` and `double` and for `String`, `Object`, and `char[ ]`. No confusion results.

4.6 R-values that cannot be L-values: literals (0, 'c', etc.); function calls; an expression, such as `x+1`. L-values: variable names (e.g., `a`), subscripted names (e.g., `x[i]`), pointer references (e.g., `*p`).

All L-values can be R-values since an address can be replaced by the value stored at that address.