Terence Henriod

Test 2 Corrections

CS 302: Data Structures

November 6, 2013

|  |  |
| --- | --- |
| The Question: | Given the declaration int & m = k;  what does the following statement do? m++;   1. Increments k 2. Increments m 3. Increments whatever k points to 4. Increments both m and k 5. Nothing – it generates a compile-time error |
|  |  |
| Correct Answer: | (D) Increments both m and k |
|  |  |
| Supporting Quote from Text: | This is a technique known as aliasing; this allows m and k to both refer to the same memory address allocated for an integer. Thus, incrementing one of the two is actually incrementing one thing with two names, so in a way, both are incremented.  “A reference variable is an alias for another variable. Any changes made to the reference variable are actually performed on the variable for which it is an alias.” |
|  |  |
| Text and Page Reference: | Starting Out With C++: From Control Structures through Objects, by Gaddis; page 351 (The text for this class does not seem to have a suitable discussion about aliasing) |

|  |  |
| --- | --- |
| The Question: | Define the **scope** of a variable. |
|  |  |
| Correct Answer: | Put simply, it is the frame of reference where the variable can be referred to by name. |
|  |  |
| Supporting Quote from Text: | “The scope of a variable is the part of the program where the variable may be used… The first rule of scope you should learn is that a variable cannot be used in any part of the program before the definition.”  “**NOTE:** When a program is running and it enters the section of code that constitutes a variable’s scope, it is said that the variable *comes into scope*. This simply means the variable is now visible and the program may reference it. Likewise, when a variable *leaves scope*, it may no longer be used.”  “When a value is stored in a static member variable, it is not stored in an instance of the class… You can think of static member variables and static member functions as belonging to the class instead of to an instance of the class.” |
|  |  |
| Text and Page Reference: | Starting Out With C++: From Control Structures through Objects, by Gaddis; pages 59, 213, and 800, respectively. (The text for this class does not seem to have a suitable discussion about scope) I was also told this in the office of Frederick C Harris, Jr. |

|  |  |
| --- | --- |
| The Question: | Define the **lifetime** of a variable. |
|  |  |
| Correct Answer: | The lifetime of a variable is from when the variable is created to when it is destroyed. |
|  |  |
| Supporting Quote from Text: | “A function s local variables exist only while the function is executing. This is known as the *lifetime* of a local variable. When the function begins, its local variables and its parameter variables are created in memory, and when the function ends, the local variables and parameter variables are destroyed.”  “Static local variables are not destroyed when a function returns. They exist for the lifetime of the program, even though their scope is only the function in which they are defined.”  “The lifetime of a variable is just from when it is created to when it is destroyed.” |
|  |  |
| Text and Page Reference: | Starting Out With C++: From Control Structures through Objects, by Gaddis; pages 338, 345 (The text for this class does not seem to have a suitable discussion about lifetime)  I was also told this in the office of Frederick C Harris, Jr. |