CS 253 Fall 2015: Things You Should Know
September 29, 2015

Disclaimer: this study guide is provided in the spirit of assisting students to organize and recall what has been covered. It is not a contract and material not appearing here may still appear on the exam.

1. You understand that in C++ variables are not automatically initialized and you also understand just how quickly and badly this can wreak havoc. (Lecture 3, Example 2).

2. You understand integer and floating-point representations in C++ along with some of the common limitations/features associated with both. (Lecture 3, Examples 3 through 8).

3. You understand the basic form of iteration supported by for loops in C++ and the associated common idioms.

4. You understand basic text output through streams such as cout and cerr.

5. You understand how precision can play a role in arithmetic using double floating point representation numbers, in particular how a number can be equal to itself plus one. (Lecture 3, Example 9).

6. You understand input streams including their creation, testing whether the next thing to read matches a declared type and how to reset terminal input. (Lecture 4, Example 1).

7. You understand one of the most common mistakes made in C++. Specifically, the use of an assignment statement in a Boolean clause associated with a conditional when one should have used the equals operator, in other words "==". (Lecture 4, example 2).

8. You understand the deceptive similarity between the legacy use of char* that is inherited from the language C and the C++ object class named "string". In particular, you'd be able to explain why the similarity is superficial and deceptive. (Lecture 4, example 3).

9. You understand, as a consumer of the added functionality, operator overloading as exemplified by string concatenation. (Lecture 4, example 4).

10. The ability to cast between compatible types is absolutely critical to the appropriate use of the language C++. The earliest examples where care must be taken with casting involve numerical types such as integer and double. You can easily produce examples upon demand of how carelessness with respect casting can lead to highly misleading if not outright incorrect results. (Lecture 4, Examples 5 and 6).

11. The ability to define functions and C++ is so utterly foundational that you may quickly begin to forget you ever learned how. That said, you are completely comfortable defining functions and also drawing the distinction between a function definition and a function prototype. And along the way, the phrase "function signature" is also meaningful to you. (Lecture 4, Examples 8 and 9).

12. Both Vectors and Arrays have their uses and their place in C++, but you would never confuse one with the other and you are comfortable now defining and using both. (Lecture 4, examples 10 and 11).

13. There are a myriad of things one might do with an object of type string and the associated ability to append an index. Just for example, you can turn a “catfish” into a “hsiftac”. (Lecture 4, Example 12).

14. When designing any function in C++ you have to take seriously how information is passed in, specifically by value or by reference, along with how you intend to make the consequences of the functions execution relevant to the outside world. For example, is it desirable to pass in an argument by reference and modify it internally knowing the changed result will be visible after the function terminates. This is sometimes called side effecting. Alternatively, is it better to return explicitly the results of what ever been changed and otherwise leave the arguments passed to the function un-altered. These trade-offs were illustrated in three simple examples. (Lecture 4, examples 12 through 14).

15. In C++, the function signature includes the types of the arguments in this further allows customization in the form of function overloading. This was illustrated in a frivolous way by showing that what a dog would say depends on the type of number shown to the dog before speaking. (Lecture 4, example 15).

16. You can tell the difference between a variable stored on the heap and one stored on the stack. Along the way, while this is seldom if every done in practice, you know how to actually inspect the address in memory where your data lives. (Lecture 7, Examples 1 through 4).

17. In C++, who says you cannot specify a pointer variable with data stored at memory address zero. Of course you can say it, you can even compile it, but you will experience a run-time error when you subsequently go to run the code. The manufactured example should make this clear, but realize that through more complex and flawed lines of code, these types of run-time errors are extremely common in C++ as programmers are gaining expertise and control of the Language. (Lecture 5, example 5).

18. In C++, it is easy to create two variables that point to the very same identical data in memory. You understand how you
could explicitly accomplish this, you also understand that in general this is going to be a very bad idea. Finally, you may even be able to think of circumstances where this is a feature rather than a bug waiting to happen. (Lecture 5, example 7)

19. You are now comfortable defining your own object classes in C++. You understand how to provide the member function prototypes as separate from the implementation. You understand how to control the visibility of member functions and member data. Last but not least, you know the difference between an object stored on the stack and an object stored in the heap. (Lecture 7, Examples 9 through 11)

20. In C++, you understand that generally it would be considered amateurish to have print statements associated with every possible object constructor and destructor. That said, as you're still learning the language, and any time you are in doubt about what is taking place, you are completely comfortable "instrumenting" your code so that you can observe explicitly and exactly when objects come into existence and when objects fade away, i.e. are destroyed. (Lecture 7, Examples 9 through 15)

21. You understand that there are actually three distinct places where you can create an object. You can name these three places. (Lecture 7, Examples 9 through 15)

22. You now understand the basic concept of "a memory leak". Knowing full well that in complex code human beings may not be excellent at spotting memory leaks, you most certainly can spot a memory leak in a relatively straightforward piece of code. (Lecture 7, example 15 and Lecture 8 Examples 1 and 2)

23. You now understand at a fundamental level the importance of stack frames, how deep they can get, which is very deep in some cases, and even the connection to language support for recursion. Along the way, you gain an understanding of how neatly objects created on the stack are automatically cleaned up as your program returns from successively more deeply nested function calls. (Lecture 8, Examples 3 through 5)

24. You understand that anywhere you always have the option to create objects on the heap, but if you do so you assume responsibility for the "You make it you break it rule". (Lecture 8, Example 6)

25. You understand the different 'flavors' of constructor commonly used including the default constructor for an object and the copy constructor. Both of which are in addition to specialized constructors you might care to create. (Lecture 9, Examples 1 through 5)

26. C++ provides a mechanism by which you can avoid needlessly constructing a default object nested within another and instead provide the necessary object is part of an initialization list. This is a mechanism you can reproduce, and fully understand. (Lecture 9, example 6)

27. In the world of object-oriented programming languages, C++ exists in the camp of languages that allow first-class status to object classes themselves. To begin, you understand broadly a statement of this kind means. Much more specifically, you understand how to associate data and functionality with an instance of the class itself as opposed to an individual instance of the class. (Lecture 9, examples 1 through 6)

28. In the language Java there is really only one way to nest and object of class foo inside an instance of an object of class bar. This is not the case in C++, which happens to be much more flexible and powerful on this particular dimension. Therefore, you are now comfortable writing code that nests objects within other objects in both the fully inclusive manner as well as the more Java like "by reference" approach. (Lecture 9, examples 7 through 9 and Lecture 10 examples 1 through 8).

29. Although programmers new to C++ may not give this functionality a great deal of thought, C++ provides for the overloading of assignment. This is a powerful and extremely useful capability. You understand now how to use this capability for new classes. (Lecture 10, Example 5)

30. Containers, data structures that allow for the representation and storage of multiple instances, are utterly essential to modern language. The container with which you are already becoming familiar is called Vector and at this point you are very comfortable using containers of objects to dynamically create and destroy object instances while also maintaining them in another object that keeps track of the entire set. (Lecture 10, Examples 9 through 11)

31. In practice, there are some very interesting internal details about how a vector of objects is expanded when new objects are added. The basic rule followed by most implementations of C++ is that the underlying memory is doubled in size before the additional object is added. After that additional object is added, the previous objects are copied from the old memory to the new. You are now comfortable using code that takes advantage of vectors of objects and can explain when and why objects are being copied as well as destroyed. (Lecture 10, Example 11)

32. Many common data structures involve objects pointing to instances of other objects of the same class. Indeed, it is even sometimes reasonable to have an object point to itself. Examples include trees, linked lists, doubly linked lists, etc. At this point you completely understand how you would go about implementing one of these data structures along with the common functionality associated with such a data structure. (Recitation for Week 6)

33. You now understand, based upon your knowledge of the stack and the heap, why it is you often must explicitly delete objects from the heap and also why you must never try to delete an object from the stack.