1. What is a *file*?
2. What does it mean to *open* a file?
3. What should a program *always* do immediately after attempting to open a file?
4. When a file has been opened, it will eventually need to be closed. But when we use C++ file streams, we mostly do not need to worry about closing files. Why not?
5. Explain how the Bubble Sort algorithm works.
6. We discussed Bubble Sort in class as an example. However, in practice, it is never used. Why not?
7. What is a *pointer*?
8. In the context of pointers, what is an *address*?
9. How do we declare a pointer in C++? (For example, write a declaration of a variable of type pointer-to-`double`.)
10. What does it mean to *dereference* a pointer?
11. Given a C++ pointer, what to we write if we wish to access the item it points to?
12. Arrays are built into C++. Nonetheless, we prefer `vector`, which is not. Why do we prefer `vector`?
13. If we have a pointer to an array item, how do we move the pointer to a different array item that is some given distance from the first?
14. What do we mean by the *distance* between two pointers?
15. How do we find the distance between two pointers in C++?
16. Fill in the blank. The *distance* between two C++ pointers should only be computed if the two pointers point to the same _____.
17. What is a *null pointer*? Why do null pointers exist?
18. What is the `const` keyword for?
19. The `const` keyword can be placed in two locations in the declaration of a pointer (e.g., "`const int * p1;`", ("`int * const p2;`"). Explain the difference between the two.
20. Your instructor says you should help the compiler find bugs for you. What does he mean? Give an example of how you can do this.
21. What do we mean by *dynamic*?
22. What is *dynamic allocation*?
23. How do we do dynamic allocation in C++?

24. When we do dynamic allocation, we must eventually do something else. What do we need to do, and how do we do it?
25. What type of value does `new` return?
26. How do we pass constructor arguments when we use `new`?
27. There are two forms of `new` (hint: one is for arrays) and two corresponding forms of `delete`. Which form of `delete` do we use with which form of `new`? Give examples of how you would write such code.
28. Explain *sequence data*, *associative data*, and *record data*.
29. What kinds of data does a `tuple` hold?
30. How do we access an item in a `tuple`?
31. What is something we can do with a C++ `tuple`, but not with a `vector`? What is something we can do with a C++ `vector`, but not with a `tuple`?
32. In C++, `vector` and `tuple` types are *first-class* types. What does this mean?
33. How do we use `enum` to create integer constants?
34. What kind of data does a `struct` hold?
35. How do we access an item in a `struct`?
36. Explain the following kinds of identifiers: *member*, *local*, *global*.
37. What is a *key-value pair*? What do key-value pairs have to do with associative data?
38. In the context of computer programming, what is a *library*?
39. What is a *namespace*?
40. Give an example of a namespace that we use often.
41. What is *namespace pollution*?
42. What is an *iterator*?
43. How do we get an iterator to the first item in a `vector`?
44. `vector` member function `end` returns an iterator. This iterator does *not* point to the last item in the `vector`. What does it point to?
45. Explain the use of the C++ keyword `auto`.
46. In C++, how do we specify a range of data using two iterators?
47. Give an example of a Standard Template Library algorithm: name it, and briefly explain what it does.
48. How do we access the items in a `pair`?
49. What kind of data does a `map` hold?
50. What is an easy way to access a value in a `map`?

51. How do we remove a key from a map?
52. What does it mean for a sorting algorithm to be *stable*?
53. What is a *lambda function*?
54. How do we create a lambda function in C++ (what is the syntax)?
55. In the context of Standard Template Library algorithms, lambda functions can be very useful. Explain.
56. What is a *pseudorandom number generator* (PRNG)?
57. What does it mean to *seed* a pseudorandom number generator?
58. What happens if we seed a pseudorandom number generator with the same value (seed) each time a program is run? Why is this behavior sometimes desirable?
59. How many times during the execution of a program do we typically seed a pseudorandom number generator?
60. Why is error checking more important when doing input than it is when doing output?
61. When we do input, we may receive valid data, or we may not. In this context, what is *valid* data?
62. How do we check for end-of-file when using a C++ input file stream?
63. Your instructor says that code should only check for end-of-file when it has already determined that there was an error on the stream. Why is this?
64. What is the difference between a *class* and an *object*?
65. How do we call a member function of a C++ object? Example:

    > Object `x` has a member function `foo` that takes no parameters. Write a single C++ statement that calls `foo` on object `x`.

66. Explain `public` vs. `private`, as these apply to members of a C++ class.
67. What does it mean when we declare a member function const?
68. How do we declare a member function const (where does the "`const`" keyword go)?
69. What is a *constructor*?
70. A constructor is a member function. What is its name?
71. What is a *default constructor?*

72. What is *overloading*?
73. C++ allows overloading of functions. What implications does this have for constructors?
74. How do we define a member function outside the class definition? (What operator do we use? Where does it go?)
75. How do we handle class-wide concerns in a C++ class (as opposed to things that concern only a particular object)?
76. What does it mean to declare a data member *static*?
77. What does it mean to declare a member function *static*?
78. How do we call a static member function in C++? Give two ways.