1. What is the C++ build process?
2. What happens in the *preprocessing* stage of the C++ build process?
3. What happens in the *compiling* step of the C++ build process?
4. What happens in the *linking* step of the C++ build process?
5. What is a *programming language*?
6. How do we print a number to the console in C++? Write the code.
7. What is the usual way to print an end-of-line in C++?
8. What does *IDE* stand for? What does it mean?
9. Most (all?) IDEs have a notion of a *project*. What is a project for?
10. How do we write a comment in C++?
11. What is a *preprocessor directive*?
12. What is an *expression*?
13. What is a *variable*?
14. What does it mean to say that a variable or expression has some particular *type*?
15. In the context of a C++ program, what is *assignment*?
16. What is a problem with the "`cin >>...`" style of input in C++?
17. What type do we use to store a string in C++? (Spell it out exactly right, including proper capitalization.) *Hint: I am **not** looking for something with "char" in it.*
18. What do we mean by *line-oriented input*?
19. What C++ Standard Library function does line-oriented input?
20. What is a `istringstream` used for?
21. What is a *condition*?
22. What is *short circuiting*?
23. What C++ comparison operator means "not equal"? Write the operator.
24. How do we check for an error on a C++ stream?
25. Write a C++ for-loop in which an integer counter variable goes from 1 to 100.
26. What does it mean to *nest* flow-of-control structures?
27. What do we mean by *flow of control*?

28. In the context of flow-of-control, what is *selection*?
29. In the context of flow-of-control, what is *iteration*?
30. When are the braces *not* necessary in a while-loop? In such cases, we often include the braces anyway. Why?
31. What C++ statement exits from a loop?
32. What C++ statement proceeds to the next iteration of a loop?
33. Explain the DRY Principle.
34. Why is the DRY Principle a good thing to follow when developing software?
35. Give a complete explanation of the meaning of the C++ keyword `break`.
36. Give a complete explanation of the meaning of the C++ keyword `continue`.
37. What is a C++ *function*?
38. What does it mean to *call* a function?
39. How can proper use of functions make our code more *DRY*?
40. What is a function *parameter*?
41. When we pass a function parameter *by value*, what happens?
42. What do we mean by the *return type* of a function?
43. What do we mean by the *scope* of an identifier?
44. What do we mean by the *lifetime* of a value?
45. What is a *local* variable.
46. What is a *side effect* of a function?
47. Are side effects a good thing? (Always, never, or sometimes?)
48. Give an example of a situation where it would be a bad idea for a function to have a side effect.
49. Name two C++ simple types.
50. What kind of data does a variable of type `bool` hold?
51. Give an example of a value that a `bool` variable can hold. Your answer should be exactly the way the value would appear in C++ source code.
52. In what context have we used `bool` values in this class, before the type `bool` was introduced by name?
53. In terms of type `bool`, what is a *condition*?

54. In each part, rewrite the given C++ code in a shorter, simpler form that still does the same thing.

```
if (b == true){
     cout << "x";
}

if (n > 3) {
    return true;
    }
else {
    return false;
    }
```

55. Write the first line of a definition of a function that has a parameter passed by value. Write the first line of a definition of a similar function that has a parameter passed by reference.
56. How does passing by reference act differently from passing by value?
57. What do we use passing by reference for?
58. What is a *data structure*?
59. What is a *container*?
60. What is a C++ `vector`?
61. Write an example of a declaration of a `vector`.
62. What is the main restriction on the types of the items in a C++ `vector`?
63. Each item in a `vector` has an *index*. What is this index? What are the legal values for an index?
64. How do we add a new item onto the end of a `vector`?
65. How can we initialize a C++ vector to a specific sequence of values, in its declaration?
66. Explain how to write a range-based for-loop.
67. When we use a range-based for-loop to iterate through the items of a vector, the loop might allow us to modify the items, or it might not. How are the two kinds of loops different? (That is, how would the C++ code you'd write be different?)
68. What is an *algorithm*?
69. What is *associative* data?

70. What is a *key-value pair*?
71. How do we lookup a single data item in an associative dataset (what information do we provide)?
72. Name the four standard single-item operations in an associative dataset.
73. Explain how the Linear Search algorithm works.
74. Explain how the Binary Search algorithm works.
75. In the context of computing, what does it mean to *sort*?
76. What does it mean for data to be *random-access*?
77. Binary Search requires a sorted dataset. In order for Binary Search to work well, the dataset must also be random-access. Linear Search requires has neither of these requirements. Why, then, do we generally prefer Binary Search to Linear Search?
78. What is a *sort*?
79. Explain how the Insertion Sort algorithm works.
80. When we name a header file, what suffix do we usually place at the end of the filename?
81. What are two things we usually put in a header file?
82. What is something we usually put in a source file?
83. What kind of data does a `char` hold?
84. How do we write a C++ character literal? How does this differ from a string literal?
85. What operator does C++ use for logical-AND?
86. What operator does C++ use for logical-OR?
87. What operator does C++ use for logical-NOT?
88. Given a simple C++ expression using logical operators, determine whether it is `true` or `false`.
    Examples.

a. `0 < 1 && 1 < 0`
b. `0 < 1 || 1 < 0`
c. `0 < 1 || 1 < 2`
d. `!true`

89. Name the primary C++ floating-point type.
90. Given a simple numeric or character literal, give its type.
    Examples.
    a. `32`
    b. `3.2`
    c. `3e2`

    d. `'e'`

91. What is a *stream manipulator*?
92. Explain the use of each of the following C++ stream manipulators.
    a. `std::fixed`
    b. `std::scientific`
    c. `std::setprecision`
    d. `std::setw`
93. What is *concatenation*?
94. How do we concatenate two strings in C++?
95. What is a *substring*?
96. How can we quickly compute a substring of a given string?
97. Explain the use of the `substr` member function of the C++ `string` type.