1. Pages 69 through 72 discuss some of the pitfalls that happen with the division operator in C++, as well as the type of expressions that involve a mix of int and double operands, and the % operator (remainder after division, sometimes called "modulus").

   a. (2 pts) What is the type of this expression: 3 + 4.5

   b. (2 pts) What is the value of this expression: 5/9

   c. (2 pts) What is the value of this expression: 5.0 / 10.0

   d. (2 pts) What is the value of this expression: 50 % 2

   e. (2 pts) What is the value of this expression: 25 % 4

2. Pages 74 through 83 discuss the if/else statement, and the C++ symbols used for the boolean operations "and" and "or".
a. (3 pts) If there is more than one statement under either the "if" part or the "else" part of an if/else in C++, what symbols do you put around the statements to group them?

b. (4 pts) Write code that will print "Bad" (followed by a newline) if the value of the variable temp is less than 50, or it is greater than 80; otherwise, it will print "Good" (followed by a newline.)

3. Section 3.1 discusses boolean expressions.

   a. (4 pts) What do we mean when we say "boolean expression"?

   b. (3 pts) What do we mean when we say "short-circuit evaluation"?

   c. (8 pts) The text discusses why we have to be careful with boolean values that convert to int values with the example

   ```
   int temperature;
   cout >> "Enter the temperature";
   cin >> temperature;
   if (!time > limit)
   // Based on what you read in that section, explain why this code can NEVER print Freezing no matter what value is entered for temperature by the user:
   ```

   ```
   int temperature;
   cout >> "Enter the temperature";
   cin >> temperature;
   if (!temperature >= 32)
   // cout << "Freezing" << endl;
4. Section 3.2 discusses multi-way branches and the "dangling else" problem, multi-way branches, and the switch statement.

   a. (2 pts) What is the dangling else problem, and how is it solved?

   b. (2 pts) What is the purpose of a "default" clause in a switch statement?

   c. (2 pts) What happens if you forget a "break" at the bottom of a "case" in a switch statement?

   d. (2 pts) In display 3.8 on p. 136, at line 28, there is a variable called "subtotal" that has an arrow pointing to it with the label "local to the block". What does this "local to the block" business mean?

   e. (3 pts) When we talk about a "block" in C++ programming, what are we talking about?

5. Section 3.3 goes into more detail that previous sections about while loops and introduces the for loop.

   a. (6 pts) A for loop header has three parts:

```cpp
  for (part1;part2;part3) {
    doSomething;
  }
```

   What are these three parts, and what does each part do?

   part1:
part2:

part3:

b. (2 pts) Typically you should NOT put a semicolon at the end of a for loop statement. But if you do, it isn't a syntax error. And, unlike Python, C++ ignores indentation (although proper indentation should STILL BE DONE for the sake of humans reading your code.) How many times will "Hi" be printed by this code (which has a semicolon where it SHOULD NOT BE)?

```cpp
for (int i = 0; i < 5; i++)
    cout << "Hi!" << endl;
```

c. (2 pts) How about now---how many times will "Hi" be printed by this same code, with the semicolon removed?

```cpp
for (int i = 0; i < 5; i++)
    cout << "Hi!" << endl;
```

6. Section 3.4 discusses the design of programs that use loops. Even though C++ may be new to you, the ideas about loops in this section should be somewhat familiar to you from previous coursework in programming, including how to use loops to accumulate a sum, how to use nested loops—even so, there may be some new and useful ideas in this section even for very experienced programmers. It is well worth reading in detail, not just for the answers to the questions below.

   a. (2 pts) When accumulating a sum, we initialize the sum variable to what value?
   b. (2 pts.) When accumulating a product, to what value would we initialize the product variable?