C++ Primer Plus, 5th *Edition* by Stephen Prata
Chapter 17: Input, Output, and Files
Review Questions

1. What role does the `iostream` file play in C++ I/O?

   The `iostream` library contains classes with which we manage input and output streams.

2. Why does typing a number such as 121 as input require a program to make a conversion?

   Typing 121 (and hitting enter) sends three bytes (corresponding to 1, 2, and 1) in succession to the program. Depending on how we choose to store the input, a conversion must be made. For example, storing 121 as an `int` requires a different byte sequence (as well as the number of bytes required) than storing it as a `double`. Of course, storing it as a string of `char` values requires the same conversion.

3. What's the difference between the standard output and the standard error?

   The standard output can be redirected to somewhere other than the moniter. The standard error cannot be redirected and will always output text to the moniter.

4. Why is `cout` able to display various C++ types without being provided explicit instructions for each type?

   I'm somewhat confused as to this question. `cout` uses template specializations to deal with various types, which are explicit instructions. If anything, output with cout may be in binary, as an `int`, as an array of `char`, or as a floating point number. If any sort of type can be converted into these basic underlying types, we can display it with `cout` by making the conversion and using the explicit instructions for displaying that basic type.

5. What feature of the output method definitions allows you to concatenate output?

   The idea that `cout` and other `ostream` classes return a reference to itself when used with the extraction operator allows us to concatenate output.

6. Write a program that requests an integer and then displays it in decimal, octal, and hexadecimal forms. Display each form on the same line, in fields that are 15 characters wide, and use the C++ number base prefixes.

   See the following code:

```
#include <iomanip>
#include <iostream>

int main()
{
    using namespace std;
    cout << "Enter an integer:  ";
    int x;
    cin >> x;
    cout << setw(15) << "decimal"
         << setw(15) << "octal"
         << setw(15) << "hexadecimal" << endl;
    cout << setw(15) << showbase << x
         << setw(15) << oct << x
         << setw(15) << hex << x << endl;
    return 0;
}
```

7. Write a program that requests the following information and that formats it as shown:

```
Enter your name: Billy Gruff
Enter your hourly wages: 12
Enter number of hours worked: 7.5
First format:
                   Billy Gruff: $    12.00: 7.5
Second format:
Billy Gruff                   : $12.00    :7.5
```

Here's the program:

```
#include<iomanip>
#include<iostream>

int main()
{
    using namespace std;
    string name;
    double wages, hours;
    cout << "Enter your name:  ";
    cin >> name;
    cout << "Enter your hourly wages:  ";
    cin >> wages;
    cout << "Enter number of hours worked:  ";
    cin >> hours;
    cout << "First format:" << endl;
    cout << right << setw(30) << name << ":  $";
    cout << setw(10) << fixed << setprecision(2) << wages;
    cout << ":" << setw(3) << setprecision(1) << hours << endl;
    cout << "Second format:" << endl;
    cout << left << setw(30) << name << ":  $";
    cout << setprecision(2) << setw(10) << wages << ":";
    cout << setprecision(1) << setw(3) << hours;
    return 0;
}
```

8. Consider the following program:

```
//rq17-8.cpp
#include <iostream>

int main()
{
    using namespace std;
    char ch;
    int ct1 = 0;

    cin >> ch;
    while (ch != 'q')
    {
        ct1++;
        cin >> ch;
    }

    int ct2 = 0;
    cin.get(ch);
    while (ch != 'q')
    {
        ct2++;
        cin.get(ch)
    }
    cout << "ct1 = " << ct1 << "; ct2 = " << ct2 << "\n";

    return 0;
}
```

What does it print, given the following input:

```
I see a q<Enter>
I see a q<Enter>
```

Here <**Enter**> signifies pressing the Enter key.

It prints the following:

```
ct1 = 5; ct2 = 9
```

This is because the extraction operator (`<<`) skips over whitespace and the `get()` method does not.

9. Both of the following statements read and discard characters up to and including the end of a line. In what way does the behavior of one differ from that of the other?

```
while (cin.get() != '\n')
    continue;
cin.ignore(80, '\n');
```

The first method will eat a line of any length. However, it won't stop at the EOF bit. The second method will eat up to a maximum of 80 characters or a newline character, whichever comes first.