C++ Primer Plus, 5^{th} *Edition* by Stephen Prata
Chapter 8: Adventures in Functions
Review Questions

1. What kinds of functions are good candidates for inline status?

   Very short functions whose function bodies consist of a single statement.

2. Suppose the `song()` function has this prototype:

   ```
   void song(char * name, int times);
   ```

   a. How would you modify the prototype so that the default value for `times` is `1`?

   For the second argument in the function signature we would replace `int times` *with* `int times = 1`.

   b. What changes would you make in the function definition?

   We could make the same changes in the function definition as we would for the prototype to include a default value for the second argument.

   c. Can you provide a default value of `"0, My Papa"` for `name`?

   Yes, provided that we also provide a default argument for `times`.

3. Write overloaded versions of `iquote()`, a function that displays its argument enclosed in double quotation marks. Write three versions: one for an `int` argument, one for a `double` argument, and one for a `string` argument.

   See the following code:

   ```
   include namespace std;

   inline void iquote(int arg) {cout << "\"" << arg << "\"\n";}
   inline void iquote(double arg) {cout << "\"" << arg << "\"\n";}
   inline void iquote(string arg) {cout << "\"" << arg << "\"\n";}

   int main(void)
   {
       ...
   }
   ```

4. The following is a structure template:

```
struct box
{
      char maker[40];
      float height;
      float width;
      float length;
      float volume;
};
```

   a. Write a function that has a reference to a `box` structure as its formal argument and displays the value of each member.

   I will list the function prototype followed by the definition:

```
void displayBox(const box & x);     // function prototype

void displayBox(const box & x)      // function definition
{
    using namespace std;
    cout << "Maker: "  << x.maker  << endl
         << "Height: " << x.height << endl
         << "Width: "  << x.width  << endl
         << "Length: " << x.length << endl
         << "Volume: " << x.volume << endl;
    return;
}
```

   b. Write a function that has a reference to a `box` structure as its formal argument and sets the `volume` member to the product of the other three dimensions.

   I will list the function prototype followed by the definition:

```
void setVolume(box & x);      // function prototype

void setVolume(box & x)       // function definition
{
    x.volume = x.height * x.width * x.length;
    return;
}
```

5. The following are some desired effects. Indicate whether each can be accomplished with default arguments, function overloading, both, or neither. Provide appropriate prototypes.

   a. `mass(density, volume)` returns the mass of an object having a density of `density` and a volume of `volume`, whereas `mass(density)` returns the mass having a density of `density` and a volume of 1.0 cubic meters. All quantities are type `double`.

   This could be accomplished with both:

   ```
   // default argument
   double mass(double density, double volume = 1.0);

   // function overloading
   double mass(double density, double volume);
   double mass(double density);
   ```

   b. `repeat(10, "I'm OK")` displays the indicated string 10 times, and `repeat("But you're kind of stupid")` displays the indicated string 5 times.

   This could be accomplished by function overloading:

   ```
   // function overloading
   void repeat(int times, const char * str);
   void repeat(const char * str);
   ```

   c. `average(3,6)` returns the `int` average of two `int` arguments, and `average(3.0, 6.0)` returns the `double` average of two `double` values.

   This could be accomplished by function overloading:

   ```
   // function overloading
   int average(int first, int second);
   double average(double first, double second);
   ```

   d. `mangle("I'm glad to meet you")` returns the character `I` or a pointer to the string `"I'm mad to meet you"`, depending on whether you assign the return value to a `char` variable or to a `char *` variable.

   This can't be accomplished with either.

6. Write a function template that returns the larger of its two arguments.

   See the following code:

```
// template prototype
template <typename Any>
Any larger(Any first, Any second);

// template definition
template <typename Any>
Any larger(Any first, Any second)
{
    return (first >= second) ? first : second;
}
```

7. Given the template of Review Question 6 and the `box` structure of Review Question 4, provide a template specialization that takes two `box` arguments and returns the one with the larger volume.

   See the following code:

```
// template specialization prototype
template <>
const box & larger(const box & first, const box & second);

// template specialization definition
template <>
const box & larger(const box & first, const box & second)
{
    if (first.volume > second.volume)
        return first;
    else
        return second;
}
```