C++ Primer Plus, 5th Edition by Stephen Prata
Chapter 7: Functions: C++'s Programming Modules
Review Questions

1. What are the three steps in using a function?

   (1) *Create the function prototype*
   (2) *Create the function definition*
   (3) *Make a call to the function*

2. Construct function prototypes that match the following descriptions:

   a. `igor()` takes no arguments and has no return value
      `void igor(void);`

   b. `tofu()` takes an `int` argument and returns a `float`
      `float tofu(int x);`

   c. `mpg()` takes two type `double` arguments and returns a `double`
      `double mpg(double x, double y);`

   d. `summation()` takes the name of a `long` array and an array size as values and returns a `long` value.
      `long summation(const long arr[], int size);`

   e. `doctor()` takes a string argument (the string is not to be modified) and returns a `double` value.
      `double doctor(const string x);`

   f. `ofcourse()` takes a `boss` structure as an argument and returns nothing.
      `void ofcourse(boss x);`

   g. `plot()` takes a pointer to a `map` structure as an argument and returns a string.
      `string plot(const map * pst);`

3. Write a function that takes three arguments: the name of an `int` array, the array size, and an `int` value. Have the function set each element of the array to the `int` value.

   See the following code:

```
void setArray(int arr[], int size, int value)
{
    for (int i = 0; i < size; i++)
        arr[i] = value;
}
```

4. Write a function that takes three arguments: a pointer to the first element of a range in an array, a pointer to the element following the end of a range in the array, and an `int` value. Have the function set each element of the array to the `int` value.

   See the following code:

   ```
   void setArray(int * start, int * end, int value)
   {
       for (int i = 0; i < (end - start) - 1; i++)
           arr[i] = value;
   }
   ```

5. Write a function that takes a `double` array name and an array size as arguments and returns the largest value in that array. Note that this function shouldn't alter the contents of the array.

   See the following code:

   ```
   double max(const double arr[], int size)
   {
       double largest = arr[0];
       for (int i = 1; i < size; i++)
           if (arr[i] > largest)
               largest = arr[i];
       return largest;
   }
   ```

6. Why don't you use the `const` qualifier for function arguments that are one of the fundamental types?

   C++ passes arguments to functions by value, so the function does not alter the value of its actual arguments. Instead, it creates new variables and initializes them to the values of the actual arguments.

7. What are the three forms a C-style string can take in a C++ program?

   (1) *As an array of* `char`
   (2) *As a pointer to a* `char`
   (3) *As a variable of type* `string`

8. Write a function that has this prototype:

   `int replace(char * str, char c1, char c2);`

   Have the function replace every occurrence of `c1` in the string `str` with `c2`, and have the function return the number of replacements it makes.

   See the following code:

```
int replace(char * str, char c1, char c2)
{
    int replacements = 0;
    for (int i = 0; i < strlen(str); i++)
    {
        if (str[i] == c1)
        {
            str[i] = c2;
            replacements++;
        }
    }
    return replacements;
}
```

9. What does the expression `*"pizza"` mean? What about `"taco"[2]`?

   `*"pizza"` *is* `'p'`. *A string literal is actually the memory address of the first character, which in this case is* `'p'`. *The dereferencing operator applied to a memory address returns the value stored at the memory address.*

   `"taco"[2]` *is* `'c'`. *We know that a string literal is actually the memory address of the first character in the string. The array notation* `"taco"[2]` *is equivalent to* `*("taco" + 2)`. *The quantity in the parentheses is the memory address of the character* `'c'`, *so applying the dereferencing operator gets us the value stored at that address.*

10. C++ enables you to pass a structure by value, and it lets you pass the address of a structure. If `glitz` is a structure variable, how would you pass it by value? How would you pass its address? What are the trade-offs of the two approaches?

    Passing `glitz` *by value is similar to passing any fundamental type by value. For the formal parameter in the function prototype and definition, we would insert* `type x` *where* `type` *would be the name of the structure type. When calling the function, the actual argument would be* `glitz`.

    To pass `glitz` *by value, we would have* `type * x` *as the formal parameter in the function, where* `type` *would be the structure type. As for the actual argument, we would use* `&glitz`.

    If we pass a structure by value, we risk wasting space and memory if the structure is large but our code writing is simpler.

11. The function `judge()` has a type `int` return value. As an argument, it takes the address of a function. The function whose address is passed, in turn, takes a pointer to a `const char` as an argument and returns an `int`. Write the function prototype.

    `int judge(int (*pf)(const char * pc));`