C++ PRIMER PLUS, 5th EDITION
PROGRAMMING EXERCISES
CHAPTER 8

1. Write a function that normally takes one argument, the address of a string, and prints that string once. However, if a second type, type `int`, argument is provided and is nonzero, the function should print the string a number of times equal to the number of times that the function has been called at that point. (Note that the number of times the string is printed is not equal to the value of the second argument, it is equal to the number of times teh function has been called.) Yes, this is a silly function, but it makes you use some of the techniques discussed in this chapter. Use the function in a simple program that demonstrates how the function works.

2. The `CandyBar` structure contains three members. The first member holds the brand name of a candy bar. The second member holds the weight (which may have a fractional value) of the candy bar, and the third member holds the number of calories (an integer value) in the candy bar. Write a program that uses a function that takes as arguments a reference to `CandyBar`, a pointer-to-`char`, a `double`, and an `int` and uses the last three values to set the corresponding members of the structure. The last three arguments should have default values of "Millennium Munch", 2.85, and 350. Also, the program should use a function that takes a reference to a `CandyBar` as an argument and displays the contents of the structure. Use `const` where appropriate.

3. Write a function that takes a reference to a `string` object as its parameter and that converts the contents of the `string` to uppercase. Use the `toupper()` function described in table 6.4. Write a program that uses a loop which allows you to test the function with different input. A sample run might look like this:

```
Enter a string (q to quit): go away
GO AWAY
Next string (q to quit): good grief!
GOOD GRIEF!
Next string (q to quit): q
Bye.
```

4. The following is a program skeleton:

```
#include <iostream>
using namespace std;
#include <cstring>
struct stringy {
    char * str;
    int ct; };

// prototypes for set(), show(), and show() go here
int main()
{
    stringy beany;
    char testing[] = "Reality isn't what it used to be.";

    set(beany, testing);     // first argument is a reference,
        // allocates space to hold copy of testing,
        // sets str member of beany to point to the
        // new block, copies testing to new block,
        // and sets ct member of beany
    show(beany);        // prints member string once
    show(beany, 2);     // prints member string twice
    testing[0] = 'D';
    testing[1] = 'u';
    show(testing);      // prints testing string once
    show(testing, 3);   // prints testing string thrice
    show("Done!");
    return 0;
}
```

Complete this skeleton by providing the described functions and prototypes. Note that there should be two `show()` functions, each using default arguments. Use `const` arguments when appropriate. Note that `set()` should use `new` to allocate sufficient space to hold the designated string. The techniques used here are similar to those used in designing and implementing classes. (You might have to alter the header filenames and delete the `using` directive, depending on your compiler.)

5. Write a template fucntion `max5()` that takes as its argument an array of five items of type `T` and returns the largest item in the array. (Because the size is fixed, it can be hard-coded into the loop instead of being passed as an argument.) Test it in a program that uses the function with an array of five `int` value and an array of five `double` values.

6. Write a template function `maxn()` that takes as its arguments an array of items of type `T` and an integer representing the number of elements in the array and that returns teh largest item in the array. Test it in a program that uses the function template with an array of six `int` values and an array of four `double` values. The program should also include a specialization that takes an array of pointers-to-`char` as an argument and the number of pointers as a second argument and that returns the address of the longest string. If multiple strings are tied for having the longest length, the function should return the address of the first one tied for longest. Test the specialization with an array of five string pointers.

7. Modify Listing 8.14 so that the template functions return the sum of the array contents instead of displaying the contents. The program should now report the total number of things and the sum of all the debts.