C++ PRIMER PLUS, 5th EDITION
PROGRAMMING EXERCISES
CHAPTER 11

1. Modify Listing 11.15 so that it writes the successive locations of the random walker in a file. Label each position with the step number. Also have the program write the initial conditions (target distance and step size) and the summarized results to the file. The file contents might look like this:

```
Target Distance:  100, Step Size:  20
0:  (x,y) = (0, 0)
1:  (x,y) = (-11.4715, 16.383)
2:  (x,y) = (-8.68807, -3.42232)
...
26:  (x,y) = (42.2919, -78.2594)
27:  (x,y) = (58.6429, -89.7309)
After 27 steps, the subject has the following location:
(x,y) = (58.6749, 089.7309)
or (m,a) = (107.212, -56.8194)
Average outward distance per step = 3.79081
```

2. Modify the `Vector` class header and implementation files (Listings 11.13 and 11.14) so that the magnitude and angle are no longer stored as data components. Instead, they should be calculated on demand when the `magval()` and `angval()` methods are called. You should leave the public interface unchanged (the same public methods with the same arguments), but alter the private section, including some of the private method, and the method implementations. Test the modified version with Listing 11.15, which should be left unchanged because the public interface of the `Vector` class is unchanged.

3. Modify Listing 11.15 so that instead of reporting the results of a single trial for a particular target/step combination, it reports the highest, lowest, and average number of steps for N trials, where N is an integer entered by the user.

4. Rewrite the final `Time` class example (Listings 11.10, 11.11, and 11.12) so that all the overloaded operators are implemented using friend functions.

5. Rewrite the `Stonewt` class (Listings 11.16 and 11.17) so that it has a state member that governs whether the object interpreted in stone form, integer pounds form, or floating-point form. Overloaded the `<<` operator to replace the `show_stn()` and `show_lbs()` methods. Overload the addition, subtraction, and multiplication operators so that one can add, subtract, and multiply `Stonewt` values. Test your class with a short program that uses all the class methods and friends.

6. Rewrite the `Stonewt` class (Listing 11.16 and 11.17) so that it overloads all six relational operators. The operators should compare the `pounds` member and return a type `bool` value. Write a program that declares an array of six `Stonewt` objects and initializes the first three objects in the array declaration. Then it should use a loop to read in values used to set the remaining three array elements. Then it should report the smallest element, the largest element, and how many elements are greater or equal to 11 stone. (The simplest approach is to create a `Stonewt` object initialized to 11 stone and to compare the other objects with that object.)

7. A complex number has two parts: a real part and an imaginary part. One way to write an imaginary number is this: (3.0, 4.0). Here 3.0 is the real part and 4.0 is the imaginary part. Suppose a = (A, Bi) and c = (C, Di). Here are some complex operations:

- Addition: a + c = (A + C, (B + D)i)
- Subtraction: a - c = (A - C, (B - D)i)
- Multiplication: a × c = (A × C - B × D, (A × D + B × C)i)
- Multiplication: (x is a real number): x × c = (x × C, x × Di)
- Conjugation: ∼a = (A, -Bi)

Define a complex class so that the following program can use it with correct results.

```
#include <iostream>
using namespace std;
#include "complex0.h" // to avoid confusion with complex.h
int main()
{
    complex a(3.0, 4.0); // initialize to (3,4i)
    complex c;
    cout << "Enter a complex number (q to quit):  \n";
    while (cin >> c) {
        cout << "c is " << c << '\n';
        cout << "complex conjugate is " << ~c << '\n';
        cout << "a is " << a << '\n';
        cout << "a + c is " << a + c << '\n';
        cout << "a - c is " << a - c << '\n';
        cout << "a * c is " << a * c << '\n';
        cout << "2 * c is " << 2 * c << '\n';
        cout << "Enter a complex number (q to quit):  '\n';
    }
    cout << "Done!\n";
    return 0;
}
```

Note that you have to overload the `<<` and `>>` operators. Many systems already have complex support in a `complex.h` header file, so use `complex0.h` to avoid conflicts. Use `const` whenver warranted.

7. (continued)
Here is a sample run of the program:

```
Enter a complex number (q to quit):
real: 10
imaginary: 12
c is (10,12i)
complex conjugate is (10,-12i)
a is (3,4i)
a + c is (13, 16i)
a - c is (-7, 8i)
a * c is (-18, 76i)
2 * c is (20, 24i)
Enter a complex number (q to quit):
real: q
Done!
```

Note that `cin >> c`, through overloading, now prompts for real and imaginary parts.