Due Thursday, 1999 Dec 02, at the beginning of class.

Reading

Read Chapter 6 of the textbook. Recursion is discussed in Chapter 12. For online documentation of the Standard Template Library, where many of the classes recently presented in class appear, see http://www.sgi.com/Technology/STL/.

Problems

When submitting a program, please submit a printed copy of the program's code prepended with comments briefly describing its input and output. Please indent your code to make it readable. See Section 2.5 of the textbook for style hints. If you desire, you can also submit examples demonstrating your program's correctness.

1. Ms. Gotta Haveit Now was very pleased with your work on the timesheet program (HW8). Last Tuesday, she attended the meeting of the Java Users Group of San Antonio and thought to herself that using a class to deal with times will simplify the code and improve its maintainability. She hires you to rewrite the code to use a time class.

   For a description of what the program should do, see the previous program description. She wants you to create a `Time` class and then use `Time` objects in the program. To improve the code's maintainability, she really likes private members, both data and functions. Also, remember how she muttered "I hate reference variables. A mark of a good programmer is to use them only when necessary." You succeeded last time in avoiding unnecessary reference variables; try to do it again.

   Hints

   - When creating the `Time` class, ask yourself what the class needs to do and what a `Time` object needs to store.
   - Some classes (including perhaps the `Time` class) do not need constructors or destructors. Writing your own constructor is useful only if you need to specify an object's state (contents) when creating it. Writing your own destructor is necessary only if some action needs to be taken when an object is destroyed; the example presented in the `checking` class was a gratuitous example.
   - Instead of trying to write a function to subtract two different times, consider writing a member function decreasing its time by the value of another `Time`'s object.

2. Write a recursive function for each of the computational tasks described below. Also write a main program to test your function, such that someone could use this program to perform a variety of tests on the function. (See the recursive examples linked from Dr. Massingill's sample programs page for examples of main programs.) *Note:* To receive credit, your function must be recursive.

Summing numbers read from a file: Write a recursive function to return the sum of integers read from a file. For example, if the file contains the following lines

```
20
10
40
30
```

the function should return 100. You can assume that a calling program has opened the file. Your function should not print anything; it should just return the sum for a calling program to print.

Scaling an array: Write a recursive function to multiply every element of an array of `doubles` by a scaling factor (another `double`). For example, if the elements of the array are

```
2.2
1.1
3.3
-1.0
-2.0
-3.0
```

and the scaling factor is -2, the function should change the elements of the array to the following:

```
-4.4
-2.2
-6.6
2.0
4.0
6.0
```

Your function should not print anything; any printing should be done by the function's caller.