Assignment 4: No Cutesy Name

Due Date: March 17, 2005, 11:59pm

Note 1: This assignment is shorter than any of the previous ones. You have until the end of dead week to turn it in, but it shouldn’t take the full 1.5 weeks.

Note 2: If you'd rather type your answers, you can grab a .doc version of this assignment from the web site. Just turn in a printout. Whether you write your answers or type them, either bring the completed assignment to the last lecture or slip it under my door (Gates B28). Don't email us your solutions!

Part 1: STL Tracing
The following lines of code use STL algorithms and methods to modify various STL containers. At each specified location, draw the contents of each container. The first one has been done for you as an example.

```cpp
vector<int> godzilla(2, 1);
godzilla[0] = 4;
godzilla.push_back(5);
```

1.1) Draw the contents of godzilla:

```
godzilla: [ 4, 1, 5 ]
```

```cpp
priority_queue<int> reaper;
reaper.push(5);
reaper.push(8);
reaper.push(5);
godzilla.push_back(reaper.top());
```

1.2) Draw the contents of godzilla and reaper:
vector<int> burning(3, 5);
godzilla.insert(godzilla.end(), burning.begin(), burning.end() - 1);

1.3) Draw the contents of godzilla:

int i = accumulate(godzilla.begin(), godzilla.end(), 3);

1.4) What is the value of i?

int puffin(int i, int j)
{
    return i + (j * 2);
}

i = accumulate(godzilla.begin() + 2, godzilla.end(), 0, puffin);

1.5) What is the value of i?

i = count_if(godzilla.begin(), godzilla.end(),
             not1(bind2nd(greater_equal<int>(), 5)));

1.6) What is the value of i?
Part 2: Short Answer

2.1) Write a templatized functor named MinAndMax that can be used as an argument to the for_each algorithm. Show how your MinAndMax can be used on a vector of ints, myVector. Your functor should have a public mMin and a public mMax that a client can access to see the results.

2.2) C++ requires that overridden virtual functions have exception specifications no less restrictive than their base class'. For example:

```cpp
class B {
public:
    virtual void Saturday() throw(Exam);
};
class D : public B {
public:
    virtual void Saturday() throw();
};
```

is valid, while

```cpp
class B {
public:
    virtual void Saturday() throw(Exam);
};
class D : public B {
public:
    virtual void Saturday() throw(Exam, Party);
};
```

is not. Why does the Standard make this requirement? Source code may provide a sufficient solution, or be used as clarification for a response, but is not necessary.
2.3) We've discussed the notion of *selective instantiation* with respect to templates. Give an example where the STL uses selective instantiation. In particular, do this in two steps:

Step 1: Show the creation of an STL object with a particular template parameter. It should compile fine. It doesn't have to actually do anything.

Step 2: Show how calling a particular method on that object would cause it to no longer compile. Briefly explain why that is.

The template parameter can be whatever type you want, including something new that you come up with, but the templatized object must be from the STL.
Part 3: Get To Know a Design Pattern

3.1) Research a design pattern that we haven't talked about in class or in the book (you can not use Singleton, Factory, Proxy, Adapter, Decorator, Chain of Responsibility, or Observer). We suggest that you visit http://www.c2.com/cgi/wiki?DesignPatterns for your research. Briefly describe the pattern and give an example of how you could apply that pattern to one of the homework assignments from this class. You should say what changes you would have to make, and what the advantages would be. If you can't think of an application to the homework, you can apply it to a possible extension to one of the assignments. An example is given below, using the Decorator pattern.

The *decorator pattern* is a technique for modifying an object's behavior at runtime. Unlike a subclass, the effects of a decorator are meant to be temporary. Decorators are used to temporarily add new functionality to an existing object.

I could have used a decorator in HW2 when implementing the Show's display() method. Instead of building the functionality of displaying a Show directly into the class, I could have created a separate class, ShowDisplayer, which could decorate a Show object by adding output capabilities. There are two advantages. First, it separates the display code from the rest of the class, making the files easier to read. Second, I can imagine having several different types of Displayers (perhaps in a hierarchy?) that are all Show decorators but display in different ways. For example, there could be an HTMLShowDisplayer.