Problem Set 3

Due before midnight on Friday, October 7, 2011.

1 Assignment Goals

1. To learn how to create a derived class.
2. To learn how to use ctors.
3. To learn how to use polymorphic pointers.
4. To apply rules for good coding style.

---

2 Problem

This problem set builds on Problem Set 2. In that problem set, you are asked to define a class `RandBit` with member function `next()` that will return the next bit in a bit sequence. `RandBit` is supposed to have two modes of operation: `normalMode` and `testMode`. In `normalMode`, the output is a biased random bit generated using a pseudorandom number with initial seed s, where the probability of 0 is p. Here, p and s are parameters that are specified when `RandBit` is instantiated. In `testMode`, the bits are fake random bits, read from a file rather than being generated by any kind of a random or pseudorandom process. Somehow, the function `RandBit::next()` has to test the mode in order to do the right thing.

2.1 Derived classes

For this problem set, you are to rewrite the code for `RandBit` to use derived classes and polymorphism to accomplish the same end. When you are done, there should be a clean separation between `RandBit` and its client(s). In particular, no explicit tests for the mode of operation should be made anywhere in your code, except for the code to create the appropriate class instance in the first place.

In greater detail, your new program should have three classes: a base class `RandBit` and two derived classes, `RandBitNormal` and `RandBitTest`. The function `RandBit::next()` should be declared to be a pure virtual function.[1] A pure virtual function is declared like a virtual function but with "`=0`" in place of the definition. Be sure to also declare a virtual destructor for the class.

The two derived classes should each define `next()` in ways that are appropriate for the mode of operation that each supports.

- `RandBitNormal` performs pseudorandom bit generation.
- `RandBitTest` returns bits read from a file.

[1] This causes `RandBit` to become an abstract base class.

2.2 Coding style

One purpose of this assignment is to improve your coding style. In particular, I am looking for clean code that makes use of polymorphic pointers and polymorphic derivation. In addition, you should rewrite/restructure your code from PS2 as necessary to follow good style guidelines.

1. As usual, your code should be well documented.

2. The isolation between modules should be made as strong as possible through appropriate use of `const` and the visibility keywords `public`, `protected`, and `private`. In general, every data member should be `private` or `protected`, and every explicit or implicit parameter to a function that in fact is not modified by the function should be labeled `const`.

3. Your program should be divided into several files to reflect the modular structure of your code. One file, `main.cpp` should contain the main program. Each class definition should be placed in a file *`name`*`.hpp` file, and the implementation of that class should be in a corresponding *`name`*`.cpp` file, where *`name`* is the class name or a shortened form that is suggestive of the class name. Two or more classes may be combined in the same `.hpp` and `.cpp` files if they are closely coupled and do not make sense in isolation. For example, it is okay to put the `RandBit`, `RandBitNormal`, and `RandBitTest` class definitions together in the same file `randbit.hpp`, and the corresponding implementations in the file `randbit.cpp`.

4. Unnecessary redundancy should be avoided, unless its use helps clarity. For example, do not write `this->x` when `x` would have the same meaning.

5. Avoid overly long function definitions. If a function is more than 10 or 20 lines long, it is probably too long and should be split into smaller parts. This is not an absolute rule but a good guideline for clean code.

6. Chapter 1 of the textbook gives other style guidelines. Follow them! If you think you need to violate these rules, check with me first.

7. Make sure your storage management is clean. Run valgrind to make sure that there are no memory errors and that all memory is freed on normal program exit.

3 Testing

Your new program should be functionally identical to the requirements for your old PS2 solution, so you should run it through the same tests and make sure that it is still working correctly.

4 Deliverables

You should submit this assignment using the `submit` script that you will find in `/c/cs427/bin/` on the Zoo. Remember to submit the following items:

1. *All* source code and header files. Even if a PS2 file has not changed, submit it again. Your PS3 submission should be self-contained without reference to previous assignments.

2. A `makefile` and any other files necessary to build your project. (If you're using Eclipse with the default settings, the makefiles will be found in the directory `Debug`.)

3. The output of several runs of your program using different seeds and different numbers of experiments.

4. Unit test programs along with test input and output that you used to verify that your program was operating correctly.

5. A brief report named `report.txt` (or any other common format such as `.pdf`) describing the design choices you made in implementing the required code extensions. You can also put any other information here that the TA should know about your program.