YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 427: Object-Oriented Programming *Handout #10*
Professor M. J. Fischer *December 4, 2018*

Problem Set 8

Due before midnight on Wednesday, December 12, 2018.

1 Introduction

This *10-point* assignment is **required** for graduate students and anyone else registered under CPSC 527. It is **optional** for students registered under CPSC 427. For those students, points earned on this assignment will offset points lost on previous homework assignments, but they will *not* apply against points lost on exams, nor will they permit anyone to earn more than 100% of the possible homework points.

The due date for this assignment is the same as that for homework assignment 7. However, they are separate assignments and should be submitted separately.

2 Problem

The goal of this assignment is to modify your solution to homework assignment 7 to gather and print additional information about the progress of the simulation.

An *inventory* of blockchains is a précis of the current set of choices of all of the agents in the population. A sample inventory from a real simulation run with 10 agents is:

```
Inventory: 5 active blockchain(s):
  1 copies of   [0,1] [1,46] [2,123] [3,255]
  1 copies of   [0,1] [1,46] [2,123] [3,271]
  6 copies of   [0,1] [1,46] [2,123]
  1 copies of   [0,1] [1,46] [2,160]
  1 copies of   [0,1] [1,46] [2,236]
```

We see that all of the agents agree on the level-0 and level-1 blocks of each chain and eight agents agree on level-2 block `[2,123]`. However, two different chains have already been forked from it, so two agents have distinct level-3 blocks, and it is unclear which will eventually win out.

Thus, an inventory is a compressed version of the current choices of the population that allows one to relatively quickly find those blocks for which consensus has already been achieved. However, finding consensus blocks is not a part of this assignment. All that is required is to create the initial inventory, to maintain it step by step during the simulation, and to print it when required (in the format shown above).

3 Programming Notes

1. You should create a class `Inventory` whose sole data member is a `std::map`.

2. To maintain the inventory, you should write functions `add()` and `sub` that add (insert) and subtract (remove) blockchains from the inventory, respectively. After each simulation step, if the agent's new and old choices differ, the new one should be added and the old one subtracted from the inventory.
3. There should be no occurrences of `new` in PS8 other than those already present in PS7. In particular, the map should be composed in `Inventory` and not be a dynamic extension. Keeping the use of dynamic storage confined to classes that can manage it such as `SPtr` and `std::map` simplies your code and makes it less likely to have hidden errors. Remember the motto: "No `new`'s is good news!"
4. You should define `Blockchain::operator<()` and `Blockchain::operator==()`. The default definitions won't work properly because they end up comparing the embedded `SPtr`'s, which will always differ in their `my_id` fields, even when one is a copy of another. For our purposes, it works to compare the serial numbers of the last block in the chain to which each `Blockchain` points.
5. The same sample call on your simulator that was provided for PS7 is given again on the Zoo in `/c/cs427/code/ps8/sample.sh`, along with the corresponding output. Note that adding the inventory to your PS7 code will change the serial numbers assigned to the blocks. This is okay.

4 Grading Rubric

Your assignment will be graded according to the scale given in Figure 1 (see below).

#	Pts.	Item
1.	1	All relevant standards from previous problem sets are followed regarding submission, identification of authorship on all files, and so forth. A well-formed `Makefile` or `makefile` is submitted that specifies compiler options `-O1 -g -Wall -std=c++17`. Running `make` successfully compiles and links the project and results in an executable file `blockchain`. Each function definition is preceded by a comment that describes clearly what it does.
2.	1	Sample input and output files are submitted that show good coverage of the parameter space, e.g., small inputs, large inputs, edge cases for the probabilities (e.g., 0.0 and 1.0) as well as reasonable intermediate cases. This is in addition to the furnished sample file.
3.	1	The program shows good style. All functions are clean and concise. Inline initializations, inline functions, and `const` are used where appropriate. Variable names are appropriate to the context. Programs are consistently indented according to the course indenting style. Each class has a separate `.hpp` file and, if needed, a separate `.cpp` file.
4.	2	The guidelines in section 3 are followed.
5.	5	All of the functionality in section 2 is correctly implemented.
	10	Total points.

Figure 1: Grading rubric.