Computer Science 1 - Program 3
Simulation
Assigned: 2/18/04
Due: 3/5/04 (Friday) at 11:55pm (WebCT time)

Objective

1. To write a program implementing the sorting techniques covered in lecture.
2. To write a program implementing the stack data structure.
3. To write a producer/consumer simulation program.

The Problem

At the CS1 Corporation we have a machine that produces objects. Each object has an identification number (an integer number) which corresponds to the order in which it was generated and a rating. The rating has been determined by a quality control machine (the producer process) and is one of "E" for excellent, "G" for good, "M" for marginal, and "D" for defective. We will assume that any of the ratings "E", "G", and "M" are equally likely to occur but that the "D" rating will occur only 10% of the time. The quality control machine will generate objects at random times and place them onto a stack. The quality assurance machine (the consumer process) removes objects from the stack at random times and does the following with each object retrieved from the stack. If the rating of the object is either "E" or "G", the object is placed into a list of quality assured products available for immediate sale. If the rating of the object is "M" the object is placed into a list of objects to be checked by another department but are not to be sold. If the rating of the object is "D", then the object is simply ignored (we'll assume it is returned for scrap and will be recycled and rebuilt).

What we want to do in this program is simulate this function of our company. To do the simulation, we'll allow these processes to execute for some period of time and see what the effect is after this period of time. A simulation cycle consists of either the production of one object, the consumption of one object, the production and consumption of one object, or neither the production nor consumption of any objects. This will be controlled by the randomness of the producer and consumer processes. For the purposes of this simulation we'll assume that during any simulation cycle, there is a 70% chance that the producer process will produce an object and a 40% chance that the consumer process will consume an object. If both events occur in a simulation cycle, then assume that the production of the new item occurs BEFORE the consumption of an item (which will be the newly produced item.)

In your implementation please adhere to the following guidelines:

1. Set the stack size to contain a maximum of 20 objects.
2. Have the user input (from the keyboard) the number of events that will occur during the simulation. For our purposes here an event is either the production or consumption of an object. Note that this does not imply that 500 objects will be

produced and 500 will consumed since both the producer and consumer processes are randomly active during a simulation cycle.

3. The producer process (the quality control machine) cannot produce an object if the stack is full. In order to implement this idea, before a simulation cycle begins, check to see if the stack is full. If it is, then automatically skip the production part of the cycle.
4. The consumer process (the quality assurance machine) cannot consume an object if the stack is empty. In order to implement this idea, right before the consumer part of the simulation cycle, check to see if the stack is empty. If it is, then automatically skip the consumer part of the cycle.
5. Your sorting must be done by either a Merge Sort or a Quick Sort. You can pick either to implement.

References

Textbook: Chapter 8. Lecture notes.

Input Format

The only value that the user will enter will be the number of simulation cycles to run. Test with small values between 10 and 20, but assume that very large values could be entered by the user. However, for setting arrays limits and such, assume that no array will need to hold more than 1000 objects.

Output Specification

When the simulation ends, we want to see the following displayed:

1. A listing of the objects in the quality assured list sorted in ascending order by their identification number.
2. A listing of the objects in the to be checked list sorted in ascending order by their identification number.
3. A count of the total number of objects rated “E”.
4. A count of the total number of objects rated “G”.
5. A count of the total number of objects rated “M”.
6. A count of all of the object that were ignored as being defective.
7. A count of the number of times the stack was full at the beginning of a simulation cycle.
8. A count of the number of times the stack was empty right before the consumer part of the simulation cycle begins.
9. A listing of all the objects left in the stack when the simulation ended. These objects should be printed so that the item that was in the stack the least amount of time should be printed last.

Output Sample

Here is a sample output from running the program when the number of simulation cycles is entered as 17 and the size of the stack is assumed to be 4. Note that this is NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given. The user input is given in *italics* while the program output is in bold.

Assume the simulation cycles for the sample execution were as follows:

1. Consume object – blocked stack is empty
2. Produce object 1:M
3. Produce object 2:E
4. Consume object 2:E
5. Produce object 3:G and Consume object 3:G
6. Produce object 4:D
7. Produce object 5:M
8. Consume object 5:M
9. Produce object 6:E
10. Produce object 7:M
11. Produce object - blocked stack is full
12. Consume object 7:M
13. Produce object 8:G
14. Consume object 8:G
15. Produce object 9:E
16. Produce object – blocked stack is full and Consume object 9:E
17. Consume object 6:E

How many simulation cycles would you like to run?
17

Quality Assurance list contains:
2:E, 3:G, 6:E, 8:G, 9:E

To be Checked list contains:
5:M, 7:M

Total of 3 "E" type objects were produced
Total of 2 "G" type objects were produced
Total of 3 "M" type objects were produced
Total of 1 "D" type objects were produced

Stack was full 3 times at beginning of simulation cycle
Stack was empty 1 time when consumer process attempted to consume object

Objects remaining in stack at end of simulation were:
1:M, 4:D

Grading Details

Due to the nature of this program, a larger portion of your grade than usual will come from your implementation of the simulation. (If you do a little math, it's pretty easy to "fake" much of the output of this assignment.) Thus, a greater weighting than in the previous two assignments will be given for adhering to the implementation specifications.

Your program will be graded upon the following criteria:

1. Adhering to the implementation specifications listed above.
2. Your algorithmic design implementing the producer/consumer processes and the constraints on their activities as described above.
3. Correctness.
4. The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
5. Compatibility to either cygwin in Windows or gcc under olympus. (If your program does not compile in either of these environments, you will get a sizable deduction from your grade.)

Restrictions

Name the file you create and turn in *simulation.c*. Although you may use other compilers, your program must compile and run using gcc or cygwin. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate.

Deliverables

A single source file named *simulation.c* turned in through WebCT by the due date and time.