Computer Science 2 Spring 2005 John Franco

# Assessment: Meeting Learning Objectives

1. The student should be able to implement in C++ *Queue* or *Stack* classes using linked lists and arrays. In the case of linked lists, the student should be able to implement standard *deletion* and *insertion* methods.

   This was tested somewhat during the oral examination given to each student. Despite going over much code in class, an attendance rate of about 80%, coding experience from eight homework assignments, a homework submission rate over 70% except for homework 6 (which was optional) and homework 8 (which many students did not complete due to lack of time), the results were disastrous. Many students have not mastered the syntax of C++ in any reasonable way and they are completely befuddled by memory management problems. In questioning students about this (during and after the oral exam), it appears at least some of the blame is due to students forming study networks (which should be encouraged, I think) and in such networks a few motivated and skillful students solve problems and pass solutions on to other students. In most cases that I have seen or been told about, the skillful students write code in the presence of others, giving them the illusion that they are learning something. In some groups, the quicker minded leader will feel content about internalizing some material and stop - the others will follow suit even though they have not achieved the same level of understanding. It seems fixing this would require changing student study practices. But, I would be reluctant to require such changes. Perhaps making strong suggestions in class would help.

   *Additional comment*: A significant number of students can explain Stack, Queue, linked list and array operations using diagrams. I have not quantified this percentage but can do this next time.

2. The student should know one sorting algorithm and be able to implement it.

   Homework number 3 asks to implement a topological sort. The student represents a given partial order using a DAG. This visualization along with a natural recursive solution to the problem helps the student understand that the complexity is on the order of the number of edges in the DAG. Homework 4 asks to incrementally sort jobs with deadlines where jobs are considered sorted if their indices in the sorted list are less than their deadlines. Homework 5 asks, in part, to use a lazy implementation of the above idea. Experiments are run on data sets of 10000 or more jobs with random deadlines to compare implementations as well as worst case and average case results. More than half the class achieved a score of 10/10 on those homeworks. Another 15% received 9/10. About 10% received 0. No changes are proposed for next time.

3. Students should know basic mathematical abstractions and should be able to implement such abstractions. The student should be able to design programs using these

abstractions.

Homework assignment 7 asks to build a C++ solution to a shortest path problem where weights of edges are 1. The student must understand something about graphs to solve it. A natural solution abstraction entails simultaneous transmission of messages from visited vertices with the first message reaching the destination carrying with it a shortest path to the origin. The typical student not only sees this right away but also eventually comes to understand this idea can be implemented using Queues. Homework 8 asks to build a logic simulator. This is another message-passing problem as the output of one gate will likely feed signals to the inputs of a multitude of gates. Again the student sees Queues as a solution facilitator. Homework 2 asks to implement a solution to the Minimum Cost Spanning Tree problem. Graphs are the abstraction used to reason a solution to this problem. Students immediately see a rough solution: order all edges by increasing cost, maintain a set of edges as a solution (make it empty intitally), add edges from the ordered list one at a time, throwing out edges that cause a cycle (we call this a redundancy). Students typically can figure this part out on their own. Implementation at this stage involves, first, the abstract notion of partitioning the set of vertices so that the invariant "two vertices are in the same subset if and only if they are connected by edges in the solution set," and second, the use of arrays and group numbers on vertices to maintain the partition from iteration to iteration. The student is shown the $O(n)$ complexity of the update operation needed for the invariant to hold (a later course revisits this and shows how to update in $O(log^*(n))$ time - incidentally, the same techniques to do this are required for the lazy implementation of the solution to Homework 5). Homeworks 4 and 5 used ordered sets. Homework 6 asks to implement a simple arithmetic expression parser and evaluator and primarily introduces a non-trivial use of Stacks. Homework 3 makes use of DAGs. Homework 6 was deemed optional and only 20% submitted solutions to Homework 8. That is not considered serious since Homework 7 was the primary tester of the use of Queues and more than 50% of the class submitted solutions to that assignment with nearly every one of the submitters receiving 9/10 or better. Statistics for Homeworks 3,4,5 were given above. About 70% of the class received a grade of 9/10 or better on Homework 2.

*Comment on evaluation*: It can not be expected that students will learn this aspect of Computer Science the first time they see it. Specifically testing for elements of this thread, especially at the implementation level, will typically show either no result or some memorized result (which, from my experience in later courses, is clearly forgotten soon after being tested). The best we can do at this level is expose students to ideas, give them lots of help to implement homework solutions, and expect them to better internalize the material in future courses. Electrical Engineers will not see those future courses so care

must be taken to emphasize the higher level (more abstract) components of the course instead of the implementation level details.

4. Students should be able to do elementary problem analysis.

   Students are shown how to do elementary problem analysis in virtually all the Homeworks (see above). These skills are examined during the oral final exam given to each student. Despite a fairly good homework submission rate, the oral results are generally disappointing. Although students can regurgitate high-level lessons from the homework fairly well, they seem at a loss when asked to think creatively about attacking a problem they have not seen in class, even though it might be similar. For example, several of the problem patterns we discussed during the quarter are amenable to Greedy solutions. But, when introducing a similar problem during the oral, students frequently fail to even consider that approach. It is not clear to me whether the problem is a deficiency in student background or talent, or whether some new teaching tool is needed. I have experimented with a few students, teaching them some things personally, and then querying them at a later date. In one case (one of the brighter students in the class) I made the same point in 8 (approx.) different meetings with the student, after which time the student still could not demonstrate mastery of the material. This is very frustrating to say the least.

5. Students should use good coding practices (use good names for variables, write invariants, etc.)

   This was not evaluated formally. Students need to improve in much more important areas first.

**Final Comment**: Right now the evaluation process seems to be set up using the assumption that a student should know forever after something that is presented in front of him or her the first time. I do not think this is a reasonable assumption. It may take one time for some students to internalize an idea and 10 times for another to internalize the same idea. One student may be quicker than another to internalize something but slower to internalize something else. Students should not be held accountable for material after a quarter. We need to find some incremental way to judge student progress.