



## Student Performance Q&A:

### 2002 AP<sup>®</sup> Computer Science AB Free-Response Questions

The following comments are provided by the Chief Reader regarding the 2002 free-response questions for AP Computer Science AB. *They are intended to assist AP workshop consultants as they develop training sessions to help teachers better prepare their students for the AP Exams.* They give an overview of each question and its performance, including typical student errors. General comments regarding the skills and content that students frequently have the most problems with are included. Some suggestions for improving student performance in these areas are also included. Consultants are encouraged to use their expertise to create strategies for teachers to improve student performance in specific areas.

#### Question 1

*What was intended by the question?*

This question tested students' ability to work with two-dimensional arrays, as represented by the `apmatrix` class. In addition, students needed to be able to work with accessing and modifying objects, instances of the `Seat` class, by using their member functions.

In part (a), students were required to scan over all objects stored in an array and count the elements conditioned on characteristics obtained from accessor methods of the objects. In part (b), students were to iterate over a single row of the array and determine whether there are a given number of empty seats in a contiguous block, using the `Seat` accessor methods to determine whether a seat is empty. If there was such a block, the index of the first seat in the block is returned.

In part (c), students use the function from part (b) to find a block of empty seats of a given size in some row. If such a block exists, then the given array of passengers was assigned to that block.

*How well did students perform?*

This question was moderately difficult for the AB students. It combined accessing a two-dimensional data structure with accessing objects using their member functions. It also included moderately difficult logic. This was the longest question on the exam as well, with three significant parts.

*What were common errors or omissions?*

Most students understood part (a). Some made mistakes on using the class abstraction, with mistakes on the accesses to the elements of the `seat` class. On part (b), some students had difficulty with the logic of scanning the one dimensional array (for a row of seats) and then

counting a contiguous block of empty seats. This could be done with either an inner loop or a counter that is reset when a seat is not empty within the main loop. There were also the same problems with the abstraction.

On part (c), students sometimes had difficulty with the indexing of two arrays that was needed in order to copy the elements from the passengers array, with index starting at 0, into the seat row, with column index starting at the first empty seat of the block found.

*Based on your experience at the AP Reading, what message would you like to send to teachers that could improve the performance of their students on the exam?*

Teachers should be sure that they cover the indexing of two array structures with different starting points, as was needed in part (c) of this question. This has been a common thread in two-dimensional array problems. Of course, students should understand the fundamental data structure and the scanning of the elements of such a structure in different orders. Part (b), finding a contiguous block of empty seats within the one dimensional row was an example of a moderately difficult algorithm that is in a difficult context in this problem. The best students should have been able to understand this algorithm.

## **Question 2**

*What was intended by the question?*

This question tested students' ability to work with the queue data type, using `apqueue`. In part (a) students were asked to write a function that appended one queue to another, both given as parameters.

Part (b) was based on a card game where, for one round, two players each played cards from a pile, modeled as a queue, such that the higher value card wins all the cards played on that round. If the cards have equal value, then they are put into a discard pile (also modeled as a queue) and two more are played, until one player wins with a higher card or because the other player has run out of cards. (If both run out at the same time, no one takes the pile and not only is the round over, but the game ends in a tie.) For part of this problem, students had to read an outline of an algorithm and then correctly implement that algorithm.

*How well did students perform?*

Students did well on this problem overall. There were several ways to implement a solution, and there were many ways to make small logical errors, even if one understood the overall algorithm. Consequently, the problem was about typical for an AB problem, with a full range of scores.

*What were common errors or omissions?*

Students would miss one of the cases for terminating the round, either entirely, or they would end the round but not move the discard pile for that case. In general, students did well with the syntax of using the `apqueue` class.

*Based on your experience at the AP Reading, what message would you like to send to teachers that could improve the performance of their students on the exam?*

Teachers should try to find interesting examples that make use of the `apqueue` (and `apstack`) data structures in the context of an interesting problem. Teachers should give students some problems in which they are given the outline of an algorithm and are required to implement that algorithm.

### Question 3

*What was intended by the question?*

This question tested students' ability to implement a moderately complex data structure, a sparse matrix, using an array of linked lists. It included two standard linked list operations within this context, accessing all elements in a linked list and adding an element to a linked list. The correct element, the initial pointer for the linked list, needed to be accessed in the array of pointers that implemented the sparse array. This was all in the context of the Marine Biology Case Study (MBCS), using this sparse array to store the fish in the Environment, instead of using `apmatrix`. In part (a), students were to scan the full array to implement the `Environment AllFish` member function. In part (b), students were to implement the `AddFish` member function which required adding an element to the sparse matrix.

*How well did students perform?*

Students did well on this question. As on the A exam, there were a large number (though not so many as on the A exam) who apparently had not studied the MBCS enough, if at all (based on the number who did not attempt the question or wrote code that was clearly off target). However, among those students that had studied it, the responses were good. It was much like a standard linked list question, with many of the same sorts of errors.

*What were common errors or omissions?*

Some students did not understand the context of the question — they had not studied the MBCS enough and made errors related to the classes. Many students worked within this context with no trouble, making typical errors for processing linked lists. One such error would be not finding the node before the insertion point when doing the insertion for part (b). A second common error would be incorrectly handling the special case when the fish needed to be added first in the list.

*Based on your experience at the AP Reading, what message would you like to send to teachers that could improve the performance of their students on the exam?*

Linked lists are a standard part of the AB curriculum. The sparse matrix used here is an interesting application of the linked list combined with the `apvector`. A similar application would be a sparse numeric matrix, where most of the entries are zero and are left out of the structure (as were the "empty fish" in this problem). Of course, the MBCS must be studied in the course and the means of storing the fish in the `Environment` class is a rich source of data structure problems (the fish could be stored in an `apmatrix`, in an unordered list, using a sequential search, in an ordered list, using binary search, in a sparse array, in a binary search tree, or in a hash table).

### Question 4

*What was intended by the question?*

This question tested students' ability to access information in a binary tree in two ways. It set up a binary tree that implemented a Huffman code tree (although this was not mentioned in the problem). In part (a), students were asked to write code that would return the word specified by a code word, a string of zeroes and ones, using the rules: branch left for a zero and right for a one. When a leaf is reached, add that letter to the word and start at the root of the tree again. With the precondition that the string defined a valid word, some error checking could be left out.

In part (b), students were asked to find the binary string that defined a given letter. This involved a recursive search through the tree such that the string of zeroes and ones that led to the given letter was returned at the end. A parameter for saving the path-so-far (called pathSoFar) was included in the function specification, although the student was not required to use it. This question resembled the question-and-answer tree on the 2000 exam.

*How well did students perform?*

Students found this question difficult. It was a bit harder than the typical tree question, which is often the hardest question on the AB exam. Most students did reasonably well on part (a), but most found part (b) difficult. Unlike similar questions in the past, however, most students at least attempted to answer part (b). The students came up with quite a variety of ways of attempting to solve the problem.

*What were common errors or omissions?*

In part (a), there were two common problems. The first was to get the check for moving right or left after the check for a leaf. This would mean that the traversal could work, but the last letter would never be reached. A second common error would be to not reset the pointer used to traverse the tree back to the root after each letter was found. Students would also make errors on indexing the search string as they traversed the tree.

In part (b), there were many different ways that students attempted to search the tree. Often students would just not have a good concept of the algorithm needed for this type of search. When students did have a reasonable attempt at the algorithm, they would often make one or more of the following mistakes: not handling the base cases of the recursion completely (both checking for the letter and checking for a leaf not containing the target letter), incorrectly assembling the string for pathSoFar (or the equivalent), incorrect recursive calls, and not including both recursive calls.

*Based on your experience at the AP Reading, what message would you like to send to teachers that could improve the performance of their students on the exam?*

With this sort of question on two of the past three exams, it certainly makes clear that tree questions that include searching for an element and recording the path to that element is an important algorithm. The search down a tree to follow a specific path is a standard tree algorithm. It is important for students to recognize when recursion is needed for a tree search, as in part (b) of this question, and when it can be done iteratively (or recursively), as in part (a). Of course, students should understand that recursion is rarely (virtually never) combined with an iterative algorithm using a while loop.