Student Performance Q&A:
2001 AP® Computer Science Free-Response Questions

The following comments are provided by the Chief Faculty Consultant regarding the 2001 free-response questions for AP Computer Science. *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.

General Comments on Exam Performance

All of the questions on the A exam, and three out of four on the AB exam, involved working with classes. This reflects the evolution of the curriculum toward an object-oriented approach to programming. Many students confused the references needed for calls to member functions. Students need to understand the following distinctions:

- Within the implementation of a member function for a class, a reference to a data member or a call of another member function of this class needs no object reference; this object is understood.
- A call to a member function of one class from the implementation of a member function of another class or a free function (a function that is not a member function of any class) needs a reference to the variable (object) of that class using the obj**.**Function notation.
- In questions that have some specifications of a class or classes, the students need to use those specifications to determine the correct object — member function relationships. They should know these relationships for the case study.

Question A1

This question involved traversals of an array (`myPumps`) of objects (of the `Pump` class).

In part (a), the student needed to traverse the array and call the `Pump` member function, `ResetGallonsSold`, for each array element. Students generally traversed the array well, but had trouble with the correct syntax and object reference for the member function.

In part (b) the student needed to traverse the array and calculate and accumulate `totalsales`, using a call to another `Pump` member function, `GallonsSold`. Pumps 0 and 1 (`myPumps[0], myPumps[1]`) used a different price and students needed to recognize that and calculate that part of the sales appropriately. They found many ways to do this correctly. Again, students often had trouble recognizing that the array `myPumps` was already available as a private data member of the class and sometimes substituted another variable. They also sometimes confused the proper object reference for the member function call.

In part (c) the student needed to make appropriate calls to the functions implemented in the two previous parts. Students generally did well on this part.

Overall, this was the easiest question on the A exam and students generally scored very well.

Question A2

This question involved the insertion of a record (a book represented as a `struct` with title, author, and age range) into a list of books (`class BookList`) in order by age range.

In part (a), the students needed to define a Boolean function `LessThan` that determined the order of books in the list, by low age, then if low ages of the range were equal, by high age. Students in general did quite well on this part, although some confused the logic of the Boolean expressions and others who used a nested if structure to cover the different cases would sometimes fail to return a value in all cases.

Part (b) asked students to write the `BookList` member function `InsertOne` that inserted a single book `struct` into the list in order. This required the understanding of a more complex algorithm than problem A1. We viewed the algorithm in three parts: finding the right location, shifting books above the inserted book up in the array, inserting the new book. The most efficient way of doing this is the insertion step of an insertion sort, where a loop works in reverse down the array, shifting each element up until the proper location for the insertion is reached, then inserting the new book. However, most students did the three parts separately, searching for the location, usually sequentially going up the array, trying to shift the books above the insertion location, and inserting.

The most common type of errors were being one off in the position or shifting, and overwriting books before shifting them. Some students confused the loop and if logic and inserted the book many times.

Students were also asked to increase the size of the array as needed. The standard way of doing this is to double the size of the array whenever the number of books stored, `myCount`, reaches the capacity of the array, `myBooks.length()`. However, few students doubled the size, almost all increased the size by one each time a book was inserted. They received partial credit as long as this was done correctly. Many students also failed to understand that the array had more positions than books in the list, the number of books being indicated by `myCount`. Part of the solution required that `myCount` be incremented when a book was inserted, which many students failed to do.

Part (c) asked students to write a member function to insert all the books from a vector parameter into the list. Most students did this correctly, or nearly so, traversing the parameter second and calling the member function `InsertOne` for each element of second.

Question A3-AB2 (Marine Biology Case Study)

The Marine Biology Case Study (MBCS) was part of the course for the first time this year. Many students did not cover the case study adequately and this is reflected by the many who did not answer this question or made only a token attempt — over 40 percent of the A students and nearly 20 percent of the AB students. Among those students who wrote solutions, the scoring distribution was similar to other questions. The question did a good job of testing understanding of how a large program works and how to make modifications to that program. Teachers need to improve their coverage of the MBCS and should integrate it throughout their course, not just cover it as a separate unit toward the end of the course. A growing body of useful material is available on the Web via the College Board Web site. (Note: the overall scores for the free response section of the exam were not much different from previous years — there is usually one especially difficult question for the A exam, and this was it this year.)

The overall problem was to make modifications to the MBCS so that the fish could age, possibly die, and breed at a specified age. The question described the changes to the `Fish` and `Environment` classes needed to accomplish this and asked the students to write three of the new functions needed.

In part (a) students were asked to write the `Environment` member function `RemoveFish` that removed a fish from the environment. `RemoveFish` is almost an inverse of the `AddFish` function already in the MBCS. Students needed to place an undefined fish at the given location in the `myWorld` matrix and needed to decrement `myFishCount`. Students generally did well on this part.

In part (b) students were to write the `Breed` member function that was called when a fish bred. There were two ways of doing this (with many variations). The one that used the given abstractions of the MBCS would use the `EmptyNeighbors` function to create a neighborhood of positions and then would loop through those positions calling the `Environment AddFish` function to put a new fish into each position.

The other approach ignored `EmptyNeighbors` and recreated its logic by checking each of the four positions (to the north, east, south, west), using the `Position` functions `North()`, etc. The position needed to be checked to ensure that it was empty, using `env.IsEmpty`, and if it was, the `Environment AddFish` function was called for that position. Students would often confuse which type of object (class) a particular member function went with. For example, many students tried to call `env.Location()`, associating the `Location` member function with the `Environment` class. Of course, `Location` is a `Fish` member function, so the call `Location()` (without an object reference) would apply it to this fish — the one calling the `Breed` function. Alternatively, the `Location` function was not needed because `Breed` could simply refer to `myPos`, the `Position` data member for this fish. Students also sometimes forgot some of the parameters to the `AddFish` function, which had a modified specification.

Part (c) required students to write the new `Fish` member function `Act` which encompassed all the changes by a fish on a given step of the simulation, including the possibility of dying, the possibility of breeding if the age was correct, and the possibility of moving, using the original `Move` function. The question on the A exam had an extra hint included in this part that indicated how to choose an event with a given probability represented as a double. The first part of the function needed to check the probability of dying and call `env.RemoveFish` if the fish died. Otherwise (and this needed to be in an else clause or otherwise protected, so a dead fish could not breed or move) the `Act` function incremented, then checked this fish's age (`myAge`) to see if it could breed. If so, the `Breed` function was called, then the `Move` function was called. Because the state of the fish had a new element, age, which was changed outside the `Move` function, the `Environment Update` function had to be called after the age was changed. For the most part students who attempted this problem did reasonably well on this part, except that practically no students called the `Update` function correctly.

Because parts (b) and (c) had quite a few details needed to get the function correct, most errors lost only ½ point (out of 9), so the grade distribution among those students who attempted the problem was reasonable.

Question A4-AB1

This question involved working with two-dimensional arrays. For the A students this was somewhat more difficult than the first two questions. For the AB students, this question

was a bit easier than the others, except for AB3. The question involved magnifying a rectangular area of a window represented using a two-dimensional array.

Part (a) of this question asked the students to write a Boolean function that checked whether a given row and column were in bounds for the window represented as an `apmatrix`. Most students did well on this part. The common errors were to leave out one or more of the four bounds that needed to be checked.

Part (b) asked students to change the values in (color) a square part of the `apmatrix` representing the window. This tested students understanding of two-dimensional indices and the use of the `InBounds` function to avoid attempted accesses to points outside the array. Most students did well on this part.

Part (c) asked students to write a function to enlarge a rectangular portion of the window. The trick here was to avoid overwriting an element of the array before it was enlarged. This could be done by making a copy of either the whole window or of the rectangular part to be enlarged, then indexing over the two two-dimensional structures at the same time to write the enlarged blocks into the `apmatrix` representing the window. The other way of doing this did not use a copy but had the indexing for drawing the enlargement run from high to low values of the rows and columns, so that no value would be overwritten before it was copied into the enlarged blocks. The indexing still had to go over two two-dimensional regions, the rectangle for copying from and the larger rectangle being changed. In both cases, the indexing over the larger rectangle had to increment by the factor of the enlargement, while incrementing by one over the rectangle copied from. In either case there were many correct ways to do the indexing and array referencing, as well as many ways to get the details wrong.

In general, students did well on this problem compared to two-dimensional array problems in the past.

Question AB3

This question was really quite different from questions on past exams. It required the use of an `apvector` of `apqueues` as an intermediate structure for carrying out a radix sort. The radix sort algorithm is straightforward and was explained in the problem (it is not a required part of the curriculum, although some teachers use it as an example). Students in general performed very well on this question. One common error was to not declare the local variable needed as a return value for each of the functions `ItemsToQueues` and `QueuesToArray`, instead coding as if there were global variables for the array of queues and for the array of items. In general, students understood the algorithms for parts (a) (putting values into queues according to the value of the current digit) and (b) (reading values out of queues back into a single array) and would lose some points for missing some of the syntax for using the structures. Students also put things together in part (c) quite well. There was a fairly large group of students who did little or nothing on this problem, indicating that there are some students who do

not adequately cover the apqueue data structure. This group was balanced by a fairly large group that got a high score for this problem.

Question AB4

This question involved working with binary trees represented using dynamically created nodes and pointers. A property called the "minheap" property was defined: the children at each node have a larger value than their parent.

In part (a) students were required to write a function, MinChild, that, given the pointer to a node, returned a pointer to the child node with the smaller value, or NULL if the given pointer was either NULL or pointed to a leaf. Most students did well on this part. Some would fail to check for one of the two special conditions, the parameter T being NULL or both child pointers being NULL.

In part (b) students were required to write a Boolean function that returned true if the given tree satisfied the minheap property throughout and false otherwise. This required a recursive function to check that all child pointers down the tree satisfied the property as well as the given pointer. Typical errors would be improper recursive calls, use of a while instead of an if structure with the recursion (or trying to traverse the tree using a while structure, which cannot be done without keeping your own stack), and confusion of the logic for the return value. In general, students did well on this part.

In part (c) students were asked to write the RemoveMin function that removed the value from the root node in a minheap and restructured the tree so as to reestablish the minheap property for the remaining values. This involved traversing the tree by moving to the MinChild node at each level and copying that value up to the parent node. This could be done by recursive calls or, as a few students did, using a while loop. However, when a leaf node was reached and had to be deleted with the pointer then set to NULL, one could not simply use the pointer returned by the MinChild function. The code needed to check whether the MinChild was the left child or the right child and assign whichever it was to NULL, as well as deleting the node. Many students either missed this subtlety of the problem or made an error in the logic for it. Consequently, relatively few students got score of 9, but then the range of scores below that showed a good spread.

Summary

Students need to be familiar with the member functions and operators available to them for the AP classes. Teachers should teach these classes early, as well as introduce other classes, to ensure that students are comfortable with writing constructors and member functions. In the AB course, students should learn to use the AP classes apstack and apqueue for implementing algorithms, as occurred in question AB3. As noted at the beginning, students need to know how to use the obj.Function() syntax for calling a member function and how to determine what the appropriate obj is for a particular

member function. They also need to know that within the definition of a member function for a given class a reference to this object of that class is understood.

It was evident from the number of responses that received no credit on the Marine Biology Case Study questions that there are a substantial number of students who do not cover the case study sufficiently. The case study is available from the College Board Web site and should be downloaded for study. It provides an excellent example of an intermediate sized program organized using several interacting classes. There is a growing body of materials that can be used for teaching the MBCS. A lot of these can be found via links from the computer science section of the College Board Web site.

The AP Marine Biology Case Study should be used throughout the course, not as a separate unit at the end. It can be used to teach or reinforce many of the concepts and language constructs in the curriculum. Many of the materials referred to above can help with this.

Students should be encouraged to use the abstractions that are provided, rather than rewriting the equivalent code.

Students should be introduced to the use and simple modification of classes early in the A course. An object-oriented approach to programming will use objects throughout the course and will put the algorithms covered in the introductory CS course in the context of classes. This approach was reflected in all the questions on the A exam this year.

What can teachers do to improve performance in specific problem areas?

Examine the canonical solutions for the 1999, 2000, and 2001 exams! These are available via the College Board Web site. The 1999 Computer Science A and AB Released Exams are also available for review and study. Note in the free-response questions that are available, there is a clear trend toward more use of defining functions in the context of classes — an object-oriented approach to programming.

Use the Marine Biology Case Study throughout the whole course! The case study is structured in two parts so that some elements of part 1 can be used very early in the A course. Both parts 1 and 2 can be used at several points throughout the A course. The case study provides good examples of using and modifying a given data structure specified as a class, a topic with which students need more work. Several AB topics, particularly analysis of algorithms and alternate data structures such as linked lists and binary trees can be introduced using the MBCS.