Advanced Placement Program®

Student Performance Q&A: 1999 AP® Computer Science A and AB Section II

The following comments are provided by the Chief Faculty Consultant regarding the 1999 free-response questions for AP Computer Science A and 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.

It has been a remarkable year for the AP Computer Science program! 1999 marked the initial delivery year of the A and AB exams in a new programming language, C++, and saw a record number of exams taken (almost 19,000 total - 12,050 A and 6,600 AB, up from 10,500 in 1998 and 11,750, the previous high, in 1997). While one might expect dramatic differences in student performance due to the language shift, this was not observed. Students made conceptual errors similar to those they made in the past and did not seem stymied at all by the new delivery language. The 1999 exam will be released in its entirety following the administration of the AP Computer Science in May 2000. In the interim, more details concerning the 1999 exam can be found online at http://www.collegeboard.org/ap/computer-science/.

Question A1: A "standard" one-dimensional array question, this question measured the student's facility with one of the most basic data structures, an array (apvector) of records (structs). In part (a), the student was asked to traverse the data structure and perform a simple computation to set the GPA field for each student. Students needed to identify the case where creditHours was zero, as this would result in a divide-by-zero error. Many students neglected to handle the zero case by failing to test appropriately but, overall, most students did very well on this question.

In part (b), the student was asked to examine individual fields of the struct and return an appropriate value. There were very few errors on this part.

In part (c), the student was asked to select the seniors from the initial array and place them in a new array. During the leaders' meetings prior to the beginning of the Reading, it was determined that this part of the question was ambiguous. No mention was made as to whether the `seniors` array was to be viewed as a "new" array containing only the new seniors, or if it already had data to which the new seniors were to be appended. It was decided that credit would be given in either case. Thus, students who failed to initialize `numSeniors` to 0 were assumed to be appending to an extant array, and students who initialized `numSeniors` to 0 were assumed to be creating a new array. This resulted in students not necessarily losing any points for failing to initialize `numSeniors` to 0 (as you would not do so if you were appending to an extant list of seniors). Of course, students needed to remain consistent in order not to lose points.

Question A2: This question, a harder one-dimensional array question, asked the students to reason about an ordered array of strings. Students needed to both understand the string abstraction and develop correct algorithms for manipulating the ordered array. The solution-space for this question was very large, making it both very interesting and very difficult to grade. In part (a), the student was required to return the "insertion position" of a given string in the ordered array. Many students checked for equality only (a standard search) instead of what was required. As is often the case with array questions, students also went beyond the valid region of the array (by looping to `wordList.length()` instead of `numWords`). In addition, many students did not correctly handle the `apstring` abstraction, attempting to use invalid comparisons (e.g., `strcmp`) instead of valid `apstring` member functions and operators (e.g., ==). In the same vein, students also resorted to a character-by-character comparison of strings, which was rarely done correctly.

In part (b), the student was asked to (potentially) insert a new word in the correct position in the ordered array, shifting elements as appropriate. Many students failed to properly exploit the `WordIndex` abstraction that part (a) afforded them in solving part (b). Another common error was to use the element at the position returned by `WordIndex` to determine whether to insert the new element. If this position was equal to `numWords`, a comparison to an array location whose contents were unknown was being performed. Finally, many students inserted the new word multiple times or had problems shifting data in the array after a correct insertion.

Question A3/AB2: This question involved reasoning about the APCS Large Integer case study. In particular, students were required to add a member function to the `BigInt` class as well as implement division. As a division question using repeated subtraction had been asked on the 1997 exam, students were constrained to use binary search in their division algorithm for part (b) of this exam. As it was explicitly prohibited to use any algorithm other than binary search, a "broader than textbook" interpretation of binary search was used to determine if a student's solution was headed down a potentially correct path and thus could be scored.

While this question was "common" to both the A and AB exam, the students taking the A exam were provided with a more specific algorithmic hint in the construction of their code on part (b) than were the students taking the AB exam (as was done on a non-case study question in 1998). In addition, AB students were asked to overload the division operator (`/`) while A students were asked to write a free function, `DivPos`.

Since the algorithm for part (a) was very tightly prescribed, most students did well. Common errors included loop bounds that were incorrect or loops that went in the wrong direction. It was acceptable to access the digits directly (instead of using abstractions such as `GetDigit` or `ChangeDigit`), but students who did so usually neglected to compensate for the fact that the digits are stored as an array of characters, not integers.

In part (b), the A students were aided by the algorithmic hint and so tended to write solutions that were closer to the mark. Most AB students seemed to have a difficult time translating the invariant into working code that moved the boundaries of the search appropriately. There were also many different errors with respect to calling the `Div2()` member function written in part (a).

Question A4/AB1: This question, as much as Question A3/AB2, was substantively different from what would have been asked in Pascal. Students were provided a context for a class, given the declaration of the class, and asked to implement a constructor and two member functions for the class. While there was a lot of reading, the intent was to provide a real-life example in which to embed an interesting programming problem.

In part (a), the student was asked to write a constructor for the Quilt class that required reading data from a file. Many students had no idea how to read data from a file under control of a loop, or how the extraction operator (>>) worked. Students and teachers are strongly encouraged to examine the canonical solution for this question to see how to properly read data from an external file. In part (b), the student was asked to supply the body of the loop for the `PlaceFlipped()` member function. Most students received full credit for this part, or were off by one in their index calculations. As it was unclear from the problem specification, and not clarified by the example provided, a solution was deemed correct if the student reflected the block about its horizontal axis, or rotated it 180 degrees.

In part (c), the student was asked to alternately place blocks and flipped blocks into a matrix that represented the entire quilt. Again, as it was unclear from the problem specification, and not clarified by the example, full credit was given to both checkerboard alternation and alternation that resulted in alternating vertical stripes if there were an even number of columns. Alternating rows was not given full credit. Most students received some credit on this part, with the most common errors being confusion between the block indices and the larger matrix indices, forgetting to declare a local matrix to return, and forgetting to return the matrix once finished.

Question AB3: This question examined the student's facility with a basic dynamically allocated data structure, the linked list. The first part tested basic linked list mechanics by asking the student to insert a new member into an existing list (that may have no members). The student needed to create a new node, assign values to the data fields of the new node, and correctly insert the new node into the (potentially) existing list. Most AB students did very well on this question. Common errors included failure to distinguish between declaring a pointer and allocating new memory, and not understanding how to use the provided constructor to accomplish the task.

In part (b), the student was asked to traverse the entire list, counting the number of nodes that matched a particular criterion. Most students performed quite well on this part, with the most common errors being running off the end of the list, and forgetting to declare a temporary pointer to traverse the list. In part (c), the student was asked to reason about traversing an array of linked lists to find a maximum, which caused some cognitive overload as students appeared to have trouble "switching gears" from one data structure (the linked list) to another (the array). The most common errors were incorrectly initializing or updating the maximum value, and printing out the incorrect information (either the wrong data or clubs that were not maximal).

Question AB4: This question examined the student's facility with binary trees, the other common dynamically allocated data structure. This was a reasonably difficult tree question and the AB students performed well, although not as well as they did on Question AB3. Part (a) was the more difficult of the two parts, as it required the student to correctly keep track of their state as they recursively traversed the tree. Most students knew to use recursion to traverse a binary tree and thus received at least some points. The most common errors included failing to properly guard the NULL case, and returning incorrect values when the name was not in the subtree.

Part (b) was much simpler than part (a) and almost all students received full credit. The most common error was a failure to correctly guard and handle the NULL case.

Are there common threads of errors?
What are the general content areas/skills that need improvement?
Are there patterns of errors that appear year after year?

- ♦ 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.

- ♦ Students should be encouraged to use abstractions that are provided. Some excellent examples of this principle are using the overloaded relational operators for apstrings (e.g., ==, <, >) instead of performing character by character comparisons and the private helper functions GetDigit, AddSigDigit, ChangeDigit, and NumDigits in the Large Integer case study. Invariably, when students try to code these themselves, they get them wrong.

- ♦ On array questions, where students are provided with an explicit number of entries to examine, they typically confuse that number with the length of the array [e.g., `numStudents` vs. `roster.length()` on Question 1, part (a)]. Students also lose points by being off by one in their loops which traverse the array, e.g., using `for (i = 0; i <= numStudents; i++)` instead of `for (i = 0; i < numStudents; i++)` on Question 1. Students should also be discouraged from arbitrarily resizing arrays. If a question indicates that there is enough room to store the data (as in Question A1 and A2), there's no need to resize. Benign resizes were not reasons to lose points on the 1999 exam, but destructive resizes did lose points.

- ♦ Work on reading from external files. Files on AP exams are well formatted, so there's almost no need to resort to strange loops or member functions to read the data. A file (infile) that contained an arbitrary number of lines of text where each line contained a name (as an apstring), an age (as an int), and a QPA (as a double) would be read as follows:

```
while (infile >> name >> age >> qpa)
{
     process name, age, and qpa
}
```

Note that no reference is made to eof, and the reads are happening as extractions in the loop guard.

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

Examine the canonical solutions for the 1999 exam! The solutions are available at http://www.cs.cmu.edu/afs/cs/usr/mjs/ftp/99-solutions.txt.