



AP[®] Computer Science AB 2000 Scoring Commentary

The materials included in these files are intended for non-commercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained herein.

These materials were produced by Educational Testing Service (ETS), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,900 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 3,500 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT[®], the PSAT/NMSQT[™], the Advanced Placement Program[®] (AP[®]), and Pacesetter[®]. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2001 by College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, and the acorn logo are registered trademarks of the College Entrance Examination Board.

**AP[®] COMPUTER SCIENCE AB
2000 SCORING COMMENTARY**

Question 1

This question, which was the same on both the A and the AB exam, required the student to implement member functions of a class whose specification was given. The problem involved a method of encryption that used a 6-by-6 two-dimensional array of characters ('A' through 'Z' and '0' through '9' in arbitrary order) as the "key" for the code. In addition, the question used a `struct Point` to encapsulate the row and column of a position in the array. Thus the student needed to understand the indexing for a two-dimensional array (`apmatrix`), the use of a `struct` to encapsulate a pair of values, and how to build and/or modify `apstrings`. This was the most difficult question for A students, but it was the easiest overall for the AB students.

In part (a), the student was asked to return the `Point` that encapsulated the coordinates in the array where a given character appeared. This required a search through all the elements of the `apmatrix`, most commonly done with a pair of nested loops. Most students handled the search well, but often lost points on the construction and return of the `Point struct`.

In part (b), the student was asked to write a member function that returned as an `apstring` the encrypted pair of characters that corresponded to a pair of characters given in an `apstring` parameter. This involved pulling the characters out of the string, using the function from part (a) to get the coordinates of each character, then, by interchanging coordinates in the specified way, getting the encrypted characters from the array and putting them into an `apstring`. The most common errors were not in accessing the `apmatrix` but were in constructing the `apstring` of two characters by incorrectly assuming some operations defined for `apstring` would work with characters. Students also often incorrectly indexed the given `apstring`, typically `pair[1]` and `pair[2]` instead of `pair[0]` and `pair[1]`. Finally, students often had difficulty with the syntax involved with getting the row and column coordinates from the `Point struct` returned by the `GetCoordinates` function written in part (a). As long as there was evidence of the coordinates used as a pair, they were eligible for the algorithm points, after losing points for incorrect `Point`.

In part (c), the students were asked to write a member function that returned as an `apstring` the encrypted version of a word given as an `apstring` parameter. They needed to traverse the given string two characters at a time, pulling out each pair of characters, using the function from part (b) to encrypt that pair, then adding that pair to the `apstring` result. Each pair could be accessed from the given word either by using the `substr` function for `apstring`, or by accessing the characters and building another two character `apstring`. Students were provided with a quick reference to all the AP classes and their member functions, so they had a quick lookup for the correct form of a call to `substr`. The most common errors in this part were with the logic of the loop control, failing to index every other character and its successor correctly, and incorrectly constructing the `apstring` result. Some students also failed to handle the last character in a word with an odd number of characters.

The difficulty in this problem seemed to come from the combination of several structures that students had to deal with together. This integration of concepts is an important aspect of programming that students should be introduced to later in the A course. It also recurs throughout the AB course.

AP[®] COMPUTER SCIENCE AB 2000 SCORING COMMENTARY

Question 2

Question AB2 examined the student's facility with two standard abstract types, stack and queue, as defined by the corresponding AP classes. The students were required to use these structures to implement a common application of stacks, translating an infix expression to a postfix expression. Since the algorithm was outlined for the students, the key to the problem was an ability to correctly use `apstack` and `apqueue`. Since the infix expression is read in order left-to-right in the algorithm and the postfix expression is created in left-to-right order, an `apqueue` was the natural structure for both the IN (const reference) parameter containing the infix expression and the OUT (reference) parameter containing the resulting postfix expression.

In part (a), students were required to overload operator `<` for the struct `Token` that contained the tokens for the expressions. Most students did this quite well, either using the order on the enumerated type `TokenType` or using an appropriate Boolean expression in an "if" construct. Errors involved leaving out a true case in the expression.

Students found part (b) more challenging. Overall, the results were good on this question, but some students had difficulty with the processing of all the `TOKENS` on the top of the stack that needed to be added to the postfix expression before pushing the current token onto the stack. Students often faltered on the logic of the inner loop needed to do this (it could also be done with only one loop, if the dequeue of the current token was protected in an appropriate "if" statement). In addition, the Boolean logic for comparing tokens was a bit convoluted because operator `<=` was the more natural one to use and it had not been provided, meaning students had to generate the equivalent expression using the operator `<` defined in part (a). Consequently, students were not severely penalized for errors in the Boolean logic if the other parts of the algorithm were correct.

This problem was different from the typical problems on past exams in that it used the standard data structures `apstack` and `apqueue` to implement a moderately complex algorithm.

**AP[®] COMPUTER SCIENCE AB
2000 SCORING COMMENTARY**

Question 3

This question examined the student's facility with a basic dynamically allocated data structure, the linked list. However, this linked list of characters (`char`) was used to reimplement the `BigInt` class. Part (a) tested basic linked list mechanics by asking the student to insert a new node into an existing list (that may be empty.) The problem was not stated this way, but was stated as reimplementing the `AddSigDigit` member function for the linked list implementation of `BigInt`. The student needed to create a new node, assign to the data fields of the new node, and correctly insert the new node into the list. In addition, the data member `myNumDigits` needed to be incremented. Most 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. The most common error was the failure to correctly convert the digit to be added from an `int` to the `char` stored in the `ListNode`.

In part (b), the student was asked to implement `GetDigit` for the linked list representation. This involved traversing the linked list the requisite number of positions to locate the required digit, then converting from a `char` to an `int` to be returned. Common errors were incorrectly locating the digit either because the count was done from the wrong end (the representation had the most significant digit at the beginning of the list, not the least) or because the count was in the correct direction but off by one. The other common error was incorrectly converting from the `char` stored in the list to the `int` to be returned. Students did well with the logic of this question.

Part (c) required students to write the destructor function, `~BigInt`, that is needed when dynamic memory allocation is used. This was simply a traversal of the list, deleting each node along the way. Most students who attempted this part did well with it. Common errors were to confuse the order of operations moving the pointer to the next node and deleting the current node, or failing to even use the second pointer needed to make this work.

**AP[®] COMPUTER SCIENCE AB
2000 SCORING COMMENTARY**

Question 4

This question examined the student's facility with binary trees, the other common dynamically allocated data structure. Students found part (a) of this question relatively easy, but part (b) quite difficult. The question was based on a tree that could be searched for advice by giving "yes" or "no" answers to questions, a variation on a common game many teachers use to teach binary trees. In this particular instance, the advice was to find a movie, given the user's preferences.

Part (a) was a typical search through a binary tree, not unlike a search in a binary search tree, but guided by the user's answers to go to the "yes" branch or the "no" branch at each step until a leaf is reached. Students did very well on this part of the question. Some used recursive calls to move down the tree, some a while loop (mixing the two usually incurred a significant penalty). Although many students made syntax errors either in the recursive calls or the shifting pointers, there was only a very small penalty if the intent of the algorithm was clear. The emphasis in grading this problem was heavily on understanding the algorithm.

Part (b) was quite difficult because it introduced an idea that many students probably had not seen: keeping track of the path through a tree to a target node that is found by a complete traversal, by pushing the nodes along the path onto a stack after the recursive calls. The concept of a complete search through a recursive traversal of the tree is common enough, but the use of a stack to record the path is unusual. Nevertheless, many students did well with this part of the question. Again, syntax errors were minimally penalized if the intent of the algorithm was clear. Common errors included incorrect use of the Boolean return value of the recursive function, pushing a string onto the stack before the recursive call instead of after, failing to return false in either the question node or leaf node case when the movie was not found, failing to check whether a node was a question node or a leaf (movie) node before getting into the recursive calls.

This was the hardest question on the AB exam. A good answer on part (a) and an attempt at part (b) could get solid intermediate credit, but it stretched the best students to get full credit for this problem.