AP® Computer Science AB
2005 Scoring Commentary

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,700 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three and a half million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Visit the College Board on the Web: www.collegeboard.com.
AP Central is the official online home for the AP Program and Pre-AP: apcentral.collegeboard.com.

Question 1

Overview

This question based on the Marine Biology Simulation Case Study focused on abstraction and inheritance. Students needed to show their understanding of the case study and its interacting classes by writing member functions for a new `Salmon` class. In part (a) students were required to override the `nextLocation` method, which selected the next location based on the salmon's age and distance from its home location (both fields of the `Salmon` class). The implementation of this method required students to call `Fish` methods to access the environment and location, and `Environment` methods to obtain the empty neighbors of the salmon. In part (b) students were required to override the `act` method to produce the desired behavior. This involved inspecting the age of the salmon and its distance from the home location, breeding or moving depending on its state, and incrementing the salmon's age.

Sample: 1A
Score: 9

In part (a) the student tests for less than mature age and then returns the `super.nextLocation()` appropriately. `emptyNeighbors()` is called correctly, not improperly modified before being tested, and then used in the distance testing process. The test itself correctly seeks a distance closer than the present location using the `distanceHome()` method and returns one if a location closer to home is found. If none is found closer than the present location, the current location is returned.

In part (b) the student again tests for less than mature age and moves normally if this evaluates to `true`. The juvenile action is defined, as is the mature action, so the distinguish/actions point was given. Then the adult's location is tested against the `homeLocation`. If the `Salmon` is "home," it attempts to breed, and if it succeeds, the `die()` method is called. Finally, as specified in the text, the `age` is incremented as the last statement executed in the method. Note that no credit was awarded for testing to be in the environment; this was not considered to be critical to the evaluation of this problem.

Sample: 1B
Score: 7

In part (a) the student conducts the proper test and returns `super.nextLocation()`. `emptyNeighbors()` is properly called, not improperly modified, and used in the distance testing process. The current location is not called or used, but an attempt is made to compare distances between the local fish (actually its neighbors) and the home location. This is a common mistake, where students implement a "min" test among the neighbors but do not include the local location, causing a loss of correctness on the last compare distance point. The student does return an empty neighbor location that could have been closer to home, so credit is given for the attempt. However, current location is not returned when a closer empty neighbor is not found.

In part (b) the solution is completely correct and all points were awarded.

Question 1 (continued)

Sample: 1C
Score: 5

In part (a) the student implements the correct age test but does not return `super.nextLocation()`, which is a fairly common error. `emptyNeighbors()` is not called and not referenced for this solution. The current location is found and used for a potential solution that seeks to find a neighbor in the direction of the `homeLocation`. This is a valid attempt that did not receive correctness credit since it cannot test for all the possible closer empty neighbors. The test meant to protect the return of an empty neighbor is not correct, so the unprotected return did not receive credit. However, the correct return of the current location received credit.

In part (b) a combined test for not being at home and being less than the mature age received credit for the age test and the "at home" test independently. Credit was not awarded for the juvenile move since it could not move in the "at home" circumstance, but it can be awarded for the mature move. There was no further credit available in the succeeding code as the student attempts to reimplement `breed()` and does not use a correct call and test of the `breed()` or the `die()` method.

Question 2

Overview

This question tested students' ability to design a data structure to meet functionality and performance specifications. A class that stored a `HashMap`, mapping postal codes to sets of cities, is provided. In part (a) students were required to identify the expected Big-Oh running time of provided methods. In parts (b) and (c) additional functionality is described, and students were asked to identify appropriate data structures, describe their organization, and justify that those structures met performance constraints. Part (d) required implementing code based on the data structure defined in part (b). While the 2004 Computer Science AB Exam also included a "design" question, the 2005 question is significantly different. Instead of designing a class hierarchy, students were required to select from known data structures to design a solution that met constraints. It combined code, written descriptions, and performance analysis.

Sample: 2A
Score: 9

In part (a) the student correctly identifies the running time for both methods.

In part (b) the student recognizes that only one additional instance variable is necessary to support storage of cities and mapping of cities to sets of codes: a `TreeMap (cityToCodeMap)`. The instance variable is correctly declared and initialized. The explanation of data organization does not mention that cities are keys and values are sets of codes, but the `addCityCodePair` update explanation does—these two sections of the answer to part (b) tend to bleed into each other, so the student was given credit if either explanation acknowledged the data organization.

In part (c) the student selects a data structure that provides $O(\log N)$ potential and correctly identifies the efficiency of their selected structure (`TreeMap` → $O(\log N)$).

In part (d) the selection of a `TreeMap` in part (b) allows the student to accomplish all of the requirements for `printAllCities` (print all cities, no duplicates, in alphabetical order, in $O(N)$) simply by iterating over the `keySet` of a `TreeMap`.

Sample: 2B
Score: 7

In part (a) the student correctly identifies the running time for both methods.

In part (b) the student identifies a data structure that supports storing city names and mappings from cities to sets of codes (the `HashMap cityToCodesMap`), earning the first 2 half points. The student earned the attempt half point for initialization/declaration, but not the correctness half point, since the instance variable is not designated as `private`. The data organization explanation acknowledges that the city names will be keys and the values will be sets of codes. The description of `addCityCodePair` is not specific enough because it does not attempt to describe how a pair is added to `cityToCodesMap`.

Question 2 (continued)

In part (c) the student selects a data structure that provides $O(\log N)$ potential and correctly identifies the efficiency of the selected structure (`HashMap` → $O(1)$).

In part (d) the implementation of `printAllCities` does print all of the cities with no duplicates, but because of the choice of a `HashMap`, the cities are not printed alphabetically. Thus the student lost the 1 point given for "cities printed in alphabetical order."

Sample: 2C
Score: 3

In part (a) the student correctly identifies the running time for `getCitiesForCode` ($O(1)$) but not for `addCityCodePair`.

In part (b) the student identifies a data structure that supports storing city names (the `List cityList`) but cannot support storing mappings from cities to sets of codes, thus losing the second half point. The variables that are declared and initialized, while not supporting the mapping, are declared and initialized correctly. The explanation of data organization indicates that the cities will be stored in `cityList` but does not describe how cities will be mapped to sets of codes. The description of changes to `addCityCodePair` indicates that the city name will be added to `citySet` but again makes no mention of city→code mapping. Furthermore, changing the name of one of the method's parameters without indicating that the name will also change in the method's existing code will mean that the original code→city mapping will not be successful.

In part (c) the student was ineligible to receive either of the half points for this section because in part (b) the student does not establish a data structure that supports mapping between cities and code sets.

In part (d) the student's response contains no references to available instance variables or parameters. The student calls `getCitiesForCode` with an unknown code parameter and then tries to add them to a local `HashSet`. Note that the use of the `Set`'s `add` method here will add the `Set` returned by `getCitiesForCode` as a single object to `totalCitySet`, rather than adding the individual members of that `Set`; to add the individual members, the method `addAll` needs to be used instead. This is not considered an attempt to iterate, traverse, and print the cities, since the source of the cities is unknown. The cities, whatever they are, are not alphabetized. And since the number of cities is unknown, so is their relation to N, so the student did not receive the point for $O(N)$ efficiency.

Question 3

Overview

This question focused on tree manipulation and recursion. An extension to the `TreeNode` class is provided, adding a parent link to the existing class. In part (a) students were required to traverse a binary tree, checking every node to make sure its parent link is correctly assigned. The question strongly suggested using recursion to traverse the tree and check each node's links. In part (b) the concept of a successor node is introduced, and an iterative algorithm for finding the successor of a node is described. Students were required to implement this algorithm, handling the three possible cases: the node has no successor; the successor is below the node in the tree; or the successor is above the node in the tree.

Sample: 3A
Score: 9

In part (a) this student uses a two-parameter helper method that takes a node and its correct parent as its two parameters. The student immediately calls the helper method, passing in the `root` node and `null` as the correct `root` parent. The first check correctly handles empty trees. This student next checks that the node's parent link matches the parent parameter, which works for all nodes including `root`. Finally, the student implements a correct recursive solution to verify the parent links for all nodes in the tree.

In part (b) the student recognizes that if `t` is the node with the maximum value in the tree, then there is no successor. The code correctly uses the `maxNode` method provided in the `Tree` class and returns `null` if there is no successor. The student also recognizes that if `t` has a right subtree, then `t`'s successor is the minimum value in the right subtree, and uses the `minNode` method effectively. Finally, the student recognizes that if there is a successor in the tree but `t` has no right subtree, then the successor will be the ancestor closest to `t` with a larger value. Unfortunately, the temporary variable introduced to look for the successor is declared to be of type `Node` rather than `TreeNode`, so the score reflects a half point usage error for a confused identifier.

Sample: 3B
Score: 5

In part (a) the student first checks that the `root`'s parent is `null`. If the tree is an empty tree, however, this check will throw an exception, so the student did not get the half point for correctly handling empty trees. In the helper method the student attempts to verify parent links by comparing the node to its children's parents. The comparison, though, is between a value and a node, not two nodes or two values. This error also affects the final correctness point because it prevents the method from returning the correct Boolean value in all cases. The student does, however, implement a correct recursive traversal to reach all nodes in the tree.

In part (b) the first test determines whether or not the value at `t` is the largest value in the subtree rooted at `t`. If the test fails, `t` must have a right subtree (and the successor must be in that subtree). At this point the method will always return `t`'s right child rather than the minimum node in `t`'s right subtree. The recursive call to `successor` with `t`'s parent earned the half point for an attempt at finding a successor above `t`, although `t` would be equal to `maxNode(t)` in that case and the code would actually return `null` rather than the successor. When there is no successor in the tree, the method correctly returns `null`.

Question 3 (continued)

Sample: 3C
Score: 4

In part (a) the student first checks that the root's parent is `null`. If the tree is an empty tree, however, this check will throw an exception, so the student did not get the half point for correctly handling empty trees. The student uses a helper method with two parameters, passing in a node and its parent. The method correctly checks that the node's parent link matches the parent parameter. The student implements a correct recursive traversal to all nodes in the tree, and the various guards in the two methods ensure that there are no null pointer exceptions. The last two recursive calls do not do anything with the Boolean return value, however, so the method does not always return the correct Boolean value.

In part (b) the student finds the minimum node in `root`'s right subtree rather than `t`'s right subtree (and without first checking that `root` has a right subtree). There is a check later for whether `t` had a right subtree, distinguishing between the cases where the successor is below `t` or above, so this code earned an attempt for finding a successor below `t`. There is no loop involving traversal of the tree above `t`, which is the prerequisite for the attempt half point for finding a successor above `t`, nor does the method correctly return `null` in all cases when there is no successor to `t` in the tree.

Question 4

Overview

This question focused on manipulating various Collections in order to execute the full expansion of an email alias. Aliases and their associated sets of addresses were stored in a provided `Map`. Part (a) involved a straightforward task, implementing a method to append the contents of a `Set` onto the end of a `Queue`. In part (b) students were required to build upon this method to perform the expansion of an email alias. This required constructing a `Queue` containing the alias and then repeatedly expanding the alias at the front of the queue and appending its associated addresses to the end. The addresses themselves had to be collected in a `Set` and returned by the method.

Sample: 4A
Score: 8

In part (a) the student correctly iterates and adds each item to the end of `q`.

In part (b) the student attempts to instantiate `Set expanded` but makes it an interface `Set` rather than a specific `HashSet` or `TreeSet`, thereby losing the half point for correctness. The student correctly enqueues `alias`, accesses all items in the queue, and dequeues correctly. Within the loop, the student correctly decides between an alias and an address and processes the addresses correctly. The student incorrectly attempts to store `alias` into `expanded`, thereby losing the store expansion half point. Since `alias` will now be in `expanded` and returned, the last half point was lost for not returning only addresses.

Sample: 4B
Score: 6

In part (a) the student correctly iterates over the set earning the first 2 points of this section. Because `items.remove(…)` returns a `boolean`, `q.enqueue` does not add `items` to `q`, thus losing the 1 point for "each item added to end of `q`." Additionally, because the `Set items` is destroyed, a 1 point usage error was applied. Notice that `enque` was interpreted as a simple spelling mistake and no penalty was applied for this error.

In part (b) the student does not attempt to instantiate any set(s) and does not instantiate the `ListQueue` correctly, thereby losing the first 1½ points. Adjusting for these errors, the student then correctly enqueues `alias`, accesses all items in `queue`, and dequeues them correctly. Within the loop, the student correctly decides between an alias and an address and processes them correctly. Finally, the student returns a set that consists only of addresses.

Sample: 4C
Score: 5

In part (a) the student attempts to loop through the `Set` but does not instantiate an iterator and thereby lost the correct loop point. The student does not `enqueue` to `q`, thus losing the "each item added to end of `q`" point.

Question 4 (continued)

In part (b) the student attempts to instantiate `Set final` but makes it an interface `Set` rather than a `HashSet` or `TreeSet`, thereby losing a half point. The student does correctly instantiate `q` as a `ListQueue`. The student, by creating `temp` and then calling `appendSetToQueue` with `temp`, correctly enqueues `alias`. The student then attempts to use recursion to loop through the process but fails to access any items of the `queue`, thereby losing the access point. After penalizing the student for this error, it was assumed this problem was fixed. The student does call `q.dequeue()`, checks `addressBook` for `key` and then recursively `expandAlias`, so the student earned these 3 half points. However, the student never stores this information anywhere, so the student lost the store expansion half point. Finally, the student adds addresses to `final`, earning 1 half point, but does not return the set, losing the last half point.