AP® Computer Science A
2003 Scoring Guidelines

The materials included in these files are intended for 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 noncommercial, face-to-face teaching purposes. This permission does not apply to any third-party copyrights contained herein. This material may not be mass distributed, 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.
### Question 1

**Part A: UpdateTuition**

<table>
<thead>
<tr>
<th>Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>loop over myColleges</td>
</tr>
<tr>
<td>+1/2</td>
<td>attempt</td>
</tr>
<tr>
<td>+1/2</td>
<td>correct (Note: OK to exit after a match is found)</td>
</tr>
<tr>
<td>+1</td>
<td>identify college in array, based on collegeName</td>
</tr>
<tr>
<td>+1/2</td>
<td>attempt</td>
</tr>
<tr>
<td>+1/2</td>
<td>correct (must be in context of loop)</td>
</tr>
<tr>
<td>+1</td>
<td>set tuition</td>
</tr>
<tr>
<td>+1/2</td>
<td>attempt (must call myColleges[k].SetTuition(...) possibly with bad syntax on array or error in parameters. SetTuition(myCollege...) does not get attempt)</td>
</tr>
<tr>
<td>+1/2</td>
<td>correct</td>
</tr>
</tbody>
</table>

**Part B: GetCollegeList**

<table>
<thead>
<tr>
<th>Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1/2</td>
<td>correctly declare result vector</td>
</tr>
<tr>
<td></td>
<td>(empty list or numeric constant size without subsequent resize loses this pt)</td>
</tr>
<tr>
<td>+1/2</td>
<td>initialize counter for colleges added to result vector</td>
</tr>
<tr>
<td>+1</td>
<td>loop over myColleges</td>
</tr>
<tr>
<td>+1/2</td>
<td>attempt (must have loop that attempts to add items to result vector)</td>
</tr>
<tr>
<td>+1/2</td>
<td>correct</td>
</tr>
<tr>
<td>+1</td>
<td>identify college to add to vector</td>
</tr>
<tr>
<td>+1/2</td>
<td>attempt (must compare some parameter with data from myColleges, possibly incorrectly)</td>
</tr>
<tr>
<td>+1</td>
<td>correct</td>
</tr>
<tr>
<td>+1</td>
<td>add identified college to result vector</td>
</tr>
<tr>
<td>+1/2</td>
<td>increment counter for matching colleges (must use correctly when inserting into result vector)</td>
</tr>
<tr>
<td>+1/2</td>
<td>final size of result vector is correct (must be in context of attempt at loop over myColleges) (must also be sufficiently large throughout)</td>
</tr>
<tr>
<td>+1/2</td>
<td>return result vector (must be in context of attempt at loop over myColleges)</td>
</tr>
</tbody>
</table>

**Usage:** -1 CollegeGroup[k] instead of myColleges[k]
Question 2

Part A: EmployeeIsEligible 3 pts

Using count method
   +2 count conditions satisfied
   +1 attempt (at least two conditions checked, possible error in comparison)
   +1 correct (count value is correct for all cases)
   +1 return correct value relative to the count

Using Boolean expression
   +1 attempt (at least two conditions checked, possible error in comparison)
   +1 partial (correct value determined for at least two different true cases)
      (must have some attempt to identify a pair) (must have an emp_)
   +1 all cases correct (except for ==, >= discrepancies)

Part B: ProcessRetirements 6 pts

   +1 check eligible
      +1/2 attempt (must have reference to an Employee)
      +1/2 correct

   +1 loop over empList
      +1/2 attempt (must have context of eligible check)
      +1/2 correct (watch for fixed loop bounds such as len)

   +1/2 place non-eligible into correct position
      +1/2 attempt
      +1 correct (extra vector method must have declaration with proper size and copy back)
      (index/count for placement of non-eligible must be correctly implemented)
      (lose this point if shift array method skips consecutive eligible employees)

   +1 adjust salary budget to account for retirements
      +1/2 correctly computed
      +1/2 stored in salaryBudget instance variable

   +1 empList has been resized correctly

   +1/2 initialize count of non-eligible employees or index for placing non-eligible employees
      (get this if there is an attempt to shift multiple array elements for each employee removed)
Question 3

Part A: HasTreasure 1 pt

+1
  +1/2 attempt (must examine a position in myGrid and some boundary)
  +1/2 correct

Part B: NumAdjacent 4 pts

+1/2 declare and initialize counter
+1 Scan 3x3 block centered at this location, except this location
  +1/2 attempt (must have nested loops or at least five individual cases)
  +1/2 correct for outer eight
+1 Check HasTreasure for each location
  +1/2 attempt (use loop indices in loop case, at least two different offsets from this location for non-loop)
  +1/2 correct
+1/2 exclude this location from count
  (may skip in scan or may count and subtract out)
+1/2 increment counter
+1/2 return counter

Part C: ComputeCounts 4 pts

+1/2 declare result matrix
+1 scan over full map
  +1/2 attempt (must have nested loops and reference map boundaries, not hard-coded constants)
  +1/2 correct
+2 fill result matrix
  +1/2 check HasTreasure
  +1/2 assign 9 in true case
  +1/2 call NumAdjacent in false case
  +1/2 assign NumAdjacent in false case
+1/2 return result matrix

Usage for part C: -1 for two or more instances of missing theMap
  -1 for using myGrid as the result matrix.

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available at apcentral.collegeboard.com.
**Question 4**

**Part A:**  
**NumAlgaeAt**  
+1 correct

**Part B:**  
**Most Algae**  
+1/2 initialize state appropriately (must include index or position, and possibly max, if used)  
+1/2 loop over 

**nbrs**  
+1 check for new max in 

**nbrs**  
+1/2 attempt (must have env.NumAlgaeAt(...), array syntax does not get attempt)  
(if env. missing, usage note below applies)  
+1/2 correct  
+1/2 update state  
+1/2 return position from 

**nbrs** determined to contain max algae

**Part C:**  
**Act**  
5 pts  
+1/2 correct check for algae  
Algae present  
+1/2 remove algae with call env.RemoveAlgae(Location(), 1)  
+1/2 reset myStepsSinceFed  
+1/2 call env.Update(Location(), *this) after myStepsSinceFed updated

No algae, do fish die?  
+1/2 check if this is third step since fed (no steps increment, no point)  
+1/2 call env.RemoveFish(Location())

Fish doesn't die  
+1/2 else (or can use returns on each part) (lose this point if update applies to removed fish)  
+1/2 call Move(env) (duplicated move logic is OK if correct)  
+1/2 increment myStepsSinceFed  
+1/2 call env.Update(Location(), *this) after myStepsSinceFed updated

Note: can use myPos instead of Location()

Usage: -1/2 for once instance of missing "env." -1 for two or more missing. (max penalty -1 for problem.)

Copyright © 2003 by College Entrance Examination Board. All rights reserved.  
Available at apcentral.collegeboard.com.
2003 General Usage

Some usage errors may be addressed specifically in rubrics with points deducted in a manner other than indicated on this sheet. The rubric takes precedence.

Usage points can only be deducted if the part where it occurs has earned credit.

A usage error that occurs once on a part when the same usage is correct two or more times can be regarded as an oversight and not penalized. If the usage error is the only instance, one of two, or occurs two or more times, then it should be penalized.

A particular usage error should be penalized only once in a problem. If it occurs on different parts of a problem, it should be deducted only once.

<table>
<thead>
<tr>
<th>Non-penalized errors</th>
<th>Minor errors (1/2 point)</th>
<th>Major errors (1 point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>case discrepancies, unless confuses identifiers</td>
<td>misspelled/confused identifier (e.g., link/next)</td>
<td>reads new values for parameters (write prompts part of this point)</td>
</tr>
<tr>
<td>missing ;'s</td>
<td>no variables declared</td>
<td>function result written to output</td>
</tr>
<tr>
<td>missing { }'s where indentation clearly conveys intent</td>
<td>void function returns a value modifying a const parameter</td>
<td>uses type or class name instead of variable identifier, for example Fish.move() instead of f.move()</td>
</tr>
<tr>
<td>default constructor called with parens, e.g., BigInt b( )</td>
<td>unnecessary cout &lt;&lt; &quot;done&quot;</td>
<td>MemberFunction(obj) instead of obj.MemberFunction( )</td>
</tr>
<tr>
<td>obj.Func instead of obj.Func( )</td>
<td>unnecessary cin (to pause)</td>
<td>param.FreeFunction( ) instead of FreeFunction(param)</td>
</tr>
<tr>
<td>loop variables used outside loop</td>
<td>no * in pointer declaration</td>
<td>Use of object reference that is incorrect or not needed, for example, use of f.move() inside member function of Fish class</td>
</tr>
<tr>
<td>[r, c] or (r)(c) instead of [r][c]</td>
<td>(r,c) instead of [r][c]</td>
<td>Use of private data when it is not accessible, instead of the appropriate accessor function</td>
</tr>
<tr>
<td>= instead of == (and vice-versa)</td>
<td>use of L-&gt;item when L.item is correct (and conversely)</td>
<td>destruction of data structure (e.g., by using root ptr for traversal)</td>
</tr>
<tr>
<td>missing ( )'s around if/while tests</td>
<td>memory leak due to unneeded node decl (may be taken twice)</td>
<td></td>
</tr>
<tr>
<td>&lt;= instead of &gt;= (and vice-versa)</td>
<td>*foo.data instead of (*foo).data</td>
<td></td>
</tr>
</tbody>
</table>

Note: Case discrepancies for identifiers fall under the "not penalized" category. However, if they result in another error, they must be penalized. Sometimes students bring this on themselves with their definition of variables. For example, if a student declares "Fish fish;", then uses Fish.move() instead of fish.move(), the one point usage deduction applies.