



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.

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 whose mission is to prepare, inspire, and connect students to college and opportunity. Founded in 1900, the association is composed of more than 4,300 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 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 equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit www.collegeboard.com

Copyright © 2003 College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Vertical Teams, APCD, Pacesetter, Pre-AP, SAT, Student Search Service, and the acorn logo are registered trademarks of the College Entrance Examination Board. AP Central is a trademark owned by the College Entrance Examination Board. PSAT/NMSQT is a registered trademark jointly owned by the College Entrance Examination Board and the National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service. Other products and services may be trademarks of their respective owners.

For the College Board's online home for AP professionals, visit AP Central at apcentral.collegeboard.com.

AP[®] COMPUTER SCIENCE A 2003 SCORING GUIDELINES

Question 1

Part A:	<code>UpdateTuition</code>	3 pts
----------------	----------------------------	--------------

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

Part B:	<code>GetCollegeList</code>	6 pts
----------------	-----------------------------	--------------

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

Usage: -1 `CollegeGroup[k]` instead of `myColleges[k]`

**AP[®] COMPUTER SCIENCE A
2003 SCORING GUIDELINES**

Question 2

Part A:	<code>EmployeeIsEligible</code>	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:	<code>ProcessRetirements</code>	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 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)

**AP[®] COMPUTER SCIENCE A
2003 SCORING GUIDELINES**

Question 3

Part A:	<code>HasTreasure</code>	1 pt
----------------	--------------------------	-------------

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

Part B:	<code>NumAdjacent</code>	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:	<code>ComputeCounts</code>	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.

**AP[®] COMPUTER SCIENCE A
2003 SCORING GUIDELINES**

Question 4

Part A:	<code>NumAlgaeAt</code>	1 pt
----------------	-------------------------	-------------

+1 correct

Part B:	<code>Most Algae</code>	3 pts
----------------	-------------------------	--------------

+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:	<code>Act</code>	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.

**AP[®] COMPUTER SCIENCE A
2003 SCORING GUIDELINES**

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.

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

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.

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