



AP Computer Science AB 2001 Scoring Guidelines

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
2001 SCORING GUIDELINES**

Question 1

Part A:	Window::IsInBounds	2 points
----------------	--------------------	-----------------

- +1 attempt (must test both row and col)
- +1 correct

Part B:	Window::ColorSquare	3 points
----------------	---------------------	-----------------

- +1 double loop over square
 - +1/2 attempt (must have two nested loops with indices or single loop with two dimensions extracted)
 - +1/2 correct
- +1 check in bounds (within loop)
 - +1/2 attempt (must have row and column parameters)
 - +1/2 correct (can assume ULrow, ULcol >= 0)
- +1 assign color value (within loop)
 - +1/2 attempt (ValAt(r, c) = val gets attempt)
 - +1/2 correct (with respect to loop bounds)

Part C:	Enlarge	4 points
----------------	---------	-----------------

- +2 double loop over rectangle
 - +1 attempt (must refer to foo.numRows and foo.numCols)
 - +1 correct (must traverse right to left or copy rectangle)
- +2 ColorSquare in context of loop
 - +1/2 apply window methods appropriately
 - +1 1/2 method invocation
 - +1/2 invocation has some parameters
 - +1 correct parameters (with respect to loop update)

**AP® Computer Science AB
2001 SCORING GUIDELINES**

Question 2

Part A:	Environment::RemoveFish	2 points
----------------	-------------------------	-----------------

- +1 replace fish with undefined fish
(emptyFish with no declaration gets point but loses ½ for usage)

- +1 decrement myFishCount
(deduct 1/2 in usage if myFishCreated is changed, but only if they get myFishCount point)

Part B:	Fish::Breed	3 points (Note: 0 points if no reasonable reference to this fish's position)
----------------	-------------	---

- +1 touch exactly 4 neighbors *of this fish* (use myPos or Location())

- +1 process empty neighbors
Using EmptyNeighbors:
 - +1/2 attempt (must have loop and call to Select(x) or nbrhood[x])
 - +1/2 correctUsing repeated checks:
 - +1/2 attempt (must have multiple calls to IsEmpty on reasonable attempt at neighbor *of this fish*)
 - +1/2 correct (call `env.IsEmpty` correctly on all touched positions)

- +1 add fish
 - +1/2 attempt (must have multiple calls to AddFish (not myWorld), where 1st param. is
 - a valid position (Exception: could be nbrhood[x])
 - an attempt at neighbor *of this fish*)
 - +1/2 correct (including: 1st param. of `env.AddFish` is a correct neighbor)

Part C:	Fish::Act	4 points
----------------	-----------	-----------------

- +1 check if this fish dies and remove fish by calling `env.RemoveFish`
 - +1/2 attempt at both (include check of random # against myProbDie; use RemoveFish)
 - +1/2 both correct (rand real # < myProbDie; <= OK; cannot decrement myFishCount)

- +1/2 Breed/Move only if fish did not die (might use `else`, return in die clause, or separate guard; must have attempt at either breeding or moving; neither can be outside of guard)

- +1/2 increment myAge

- +1 breed correctly
 - +1/2 attempt at both (check age against reasonable age **AND** call `Breed`, must not reimplement `Breed`)
 - +1/2 both correct (myAge must be correct, >= 3 is not correct)

- +1/2 move (with or without else)

- +1/2 correct update (must come after age increment; must not be called for a dead fish)

Note: age vs myAge and pos vs myPos are confused identifiers (usage)

**AP[®] Computer Science AB
2001 SCORING GUIDELINES**

Question 3

Part A:	ItemsToQueues	3 points
----------------	---------------	-----------------

- +1 declare and return `apvector<apqueue<int> >`
 - +1/2 declaration (must be properly sized)
 - +1/2 return variable used in body

- +1/2 loop over `L` correctly

- +1 1/2 store item in correct queue
 - +1/2 correct index for queue based on data from vector `L`
 - +1/2 attempt to enqueue data from vector `L`
 - +1/2 correct

Part B:	QueuesToArray	4 points
----------------	---------------	-----------------

- +1 declare and return `apvector<int>`
 - +1/2 declaration (must be properly sized)
 - +1/2 return variable used in body

- +1 iteration over vector of queues
 - Note: loop may be over `numVals`, accessing appropriate queues
 - +1/2 attempt (must reference `QA` as a vector, in context)
 - +1/2 correct

- +1 iteration within each queue until empty
 - +1/2 attempt (iterate over queue)
 - +1/2 correct

- +1 copy each item to vector and remove from queue
 - +1/2 attempt (must either copy item from `QA` to vector OR remove from queue)
 - +1/2 correct

Part C:	RadixSort	2 points
----------------	-----------	-----------------

- +1/2 declare local `apvector<apqueue<int> >` OR nested function calls

- +1/2 loop over correct number of digits

- +1 for each digit, from least significant digit to most significant digit: carry out the two steps
 - +1/2 attempt (must attempt both transfer of items into queues and items from queues to vector, both inside loop)
 - does not receive this half point for reproducing function code unless it is correct
 - +1/2 correct

**AP[®] Computer Science AB
2001 SCORING GUIDELINES**

Question 4

Part A:	MinChild	3 points
----------------	----------	-----------------

- +1/2 T is NULL or has zero children

- +1 T has one child
 - +1/2 attempt (including at least one return)
 - +1/2 correct

- +1 1/2 T has two children
 - +1/2 attempt to check for smaller
 - +1 correct

Part B:	IsHeapOrdered	2 points
----------------	---------------	-----------------

- +1/2 base case (T is NULL or has zero children; i.e., MinChild(T) == NULL)

- +1 1/2 return correct value
 - +1/2 attempt to test heap condition at this node
 - +1/2 attempt at least one recursive call
(must include an argument and use the returned value)
 - +1/2 correct

Part C:	RemoveMin	4 points
----------------	-----------	-----------------

- +1 leaf case
 - +1/2 attempt (Must test condition correctly and
either deallocate memory or reset a pointer in the tree)
 - +1/2 correct

- +3 non-leaf case
 - +1/2 descends correct branch
 - +1 assign T->val
 - +1/2 attempt at recursive call (must include an argument in call to RemoveMin)
 - +1 correct (must set correct pointer in tree)

AP[®] Computer Science AB 2001 SCORING GUIDELINES

Usage Sheet

In general, no usage points are deducted for usage mistakes for which evidence of understanding appears elsewhere in the problem. For example, if there are *no* variables declared in a problem, then usage points may be deducted. However, a missing declaration in the presence of other declarations does NOT lose points. Also, we should not take off usage points for syntactically correct code that goes beyond the AP subset (e.g., using `printf` or `scanf`, or returning `0` instead of `false` for a `bool`).

Usage points can only be deducted if the PART has earned credit. Some usage errors may be addressed specifically in rubrics with points deducted in a manner other than indicated on this sheet.

Non-penalized errors

case discrepancies, unless
confuses identifiers

missing `;`'s

missing `{ }`'s where indentation
clearly conveys intent

default constructor called with
parens, e.g., `BigInt b()`

`obj.Func` instead of `obj.Func()`

loop variables used outside loop

`[r, c]` instead of `[r][c]`

`=` instead of `==` (and vice-versa)

missing `()`'s around `if/while` tests

`<<` instead of `>>` (and vice-versa)

`*foo.data` instead of `(*foo).data`

Minor errors (1/2 point)

misspelled/confused identifier
(e.g., `link/next`)

no variables declared

`MemberFunction(obj)` instead
of `obj.MemberFunction()`

`param.FreeFunction()` instead
of `FreeFunction(param)`

void function returns a value

modifying a `const` parameter

unnecessary `cout << "done"`

unnecessary `cin` (to pause)

no `*` in pointer declaration

Major errors (1 point)

reads new values for parameters
(write prompts part of this point)

function result written to output

type error (uses type name instead
of variable identifier)