AP Computer Science A
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.
Question 1

Part A: ResetAll  2 points

+1  loop over pumps
+1/2 attempt — must have some reference to pumps
+1/2 correct
+1  reset each pump
+1/2 attempt — must attempt indexing
+1/2 correct

Part B: TotalSales  5 points

+1  initialize/return total
+1/2 init — correct (int total, reset in loop lose this)
+1/2 return (premature return, cout lose this)
+2  handle full service pumps
+1  attempt — must have two different actions
+1  correct (watch for * 1.25)
+2  general case loop
+1  loop
+1/2 attempt
+1/2 correct
+1  accumulate pump sales
+1/2 attempt — must use indexing here (gallons only okay)
+1/2 correct (premature return and gallons only loses this)

Part C: CloseStation  2 points

+1  print total to logFile (text, formatting, or lack thereof OK)
+1/2 attempt (total sales OR logFile)
+1/2 correct
+1  ResetAll()
+1/2 attempt
+1/2 correct

Usage

-1  Pump/myPumps/p/Station confusion

-0  obj.Func instead of obj.Func()
## Question 2

### Part A: LessThan 2 points

+1 attempt (must have return and attempt multiple relevant age comparisons)
  
  (||, &amp; confusion OK, inverted order OK)

+1 correct

### Part B: InsertOne 5 points

+1 resize myList
  +1/2 attempt (must call resize OR construct temp vector larger than myList)
  +1/2 correct (must double size within context of if)

+1 find location (no point if LessThan is reimplemented incorrectly)
  +1/2 attempt (call to LessThan in context of if or loop)
  +1/2 correct

+1 shift items
  +1/2 attempt
  +1/2 correct (can be earned despite incorrect location index)

+1 insert at correct location
  +1/2 attempt (myList[?] = bk;)
  +1/2 correct (must have earned find/correct to get this)

+1 increment myCount exactly once

Note on sort solutions (add element at end, then sort): incorrect sort loses 'correct' on find, shift, and insert.

Note on temp vector solutions: failure to copy back to myList loses insert/correct and resize/correct unless myList is explicitly resized.

### Part C: InsertMany 2 points

+1 process all elements of second
  +1/2 attempt (loop bounds or body must mention second)
  +1/2 correct (bad call to InsertOne OK)

+1 correct call to InsertOne
Question 3

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 correct
Using 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)
Question 4

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)
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.

<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 no variables declared</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>confuses identifiers</td>
<td>no variables declared</td>
<td>function result written to output</td>
</tr>
<tr>
<td>missing 's</td>
<td>MemberFunction(obj) instead of obj.MemberFunction()</td>
<td>type error (uses type name instead of variable identifier)</td>
</tr>
<tr>
<td>missing { }'s where indentation clearly conveys intent</td>
<td>default constructor called with parens, e.g., BigInt b( )</td>
<td></td>
</tr>
<tr>
<td>missing { }'s where indentation clearly conveys intent</td>
<td>MemberFunction(obj) instead of obj.MemberFunction()</td>
<td></td>
</tr>
<tr>
<td>obj.Func instead of obj.Func( )</td>
<td>default constructor called with parens, e.g., BigInt b( )</td>
<td>void function returns a value</td>
</tr>
<tr>
<td>loop variables used outside loop</td>
<td>unnecessary cout &lt;&lt; &quot;done&quot;</td>
<td></td>
</tr>
<tr>
<td>[r, c] instead of [r][c]</td>
<td>unnecessary cin (to pause)</td>
<td></td>
</tr>
<tr>
<td>= instead of == (and vice-versa)</td>
<td>unnecessary cin (to pause)</td>
<td></td>
</tr>
<tr>
<td>missing ( )'s around if/while tests</td>
<td>no * in pointer declaration</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt; instead of &gt;&gt; (and vice-versa)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*foo.data instead of (*foo).data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>