AP® Computer Science A
2009 Scoring Guidelines

AP® COMPUTER SCIENCE A
2009 SCORING GUIDELINES

Question 1: Number Cube

Part (a)	`getCubeTosses`	**4 points**

+1 constructs array
- **+1/2** constructs an array of type `int` **or** size `numTosses`
- **+1/2** constructs an array of type `int` **and** size `numTosses`

+2 1/2 processes tosses
- **+1** repeats execution of statements `numTosses` times
- **+1** tosses cube in context of iteration
- **+1/2** collects results of tosses

+1/2 returns array of generated results

Part (b)	`getLongestRun`	**5 points**

+1 iterates over `values`
- **+1/2** accesses element of `values` in context of iteration
- **+1/2** accesses all elements of `values`, no out-of-bounds access potential

+1 determines existence of run of consecutive elements
- **+1/2** comparison involving an element of `values`
- **+1/2** comparison of consecutive elements of `values`

+1 always determines length of at least one run of consecutive elements

+1 identifies maximum length run based on all runs

+1 return value
- **+1/2** returns starting index of identified maximum length run
- **+1/2** returns -1 if no run identified

AP® COMPUTER SCIENCE A
2009 SCORING GUIDELINES

Question 2: Stockpile Critter (GridWorld)

+1 class header
- **+1/2** properly formed class header for `StockpileCritter`
- **+1/2** extends `Critter` class

+1 1/2 stockpile state
- **+1/2** declares instance variable capable of maintaining state
- **+1/2** private visibility
- **+1/2** initialization of state appropriate to usage of variable

+1 overrides methods and maintains all necessary postconditions
(No points awarded if overrides `act` *method)*

+1 `processActors` overridden *(No points awarded if overrides* `act` *method)*

+1 stockpile state maintenance
- **+1/2** accumulates based on number of actors passed to `processActors`
- **+1/2** decrements appropriately each `act`

+1 1/2 removes neighboring actors from grid
- **+1/2** removes **at least one** neighboring actor from grid
- **+1** removes **all** neighboring actors from grid

+2 self-removal
- **+1/2** checks status of stockpile by using state variable in a relational expression
- **+1/2** ever removes self from grid
- **+1** removes self from grid when and only when stockpile state indicates empty

AP® COMPUTER SCIENCE A 2009 SCORING GUIDELINES

Question 3: Battery Charger

Part (a)	`getChargingCost`	**5 points**

+1 1/2 accesses array elements
- **+1/2** accesses any element of `rateTable`
- **+1/2** accesses an element of `rateTable` using an index derived from `startHour`
- **+1/2** accesses multiple elements of `rateTable` with no out-of-bounds access potential

+2 1/2 accumulates values
- **+1/2** declares and initializes an accumulator
- **+1/2** accumulates values from elements of `rateTable`
- **+1/2** selects values from `rateTable` using an index derived from `startHour` and `chargeTime`
- **+1** determines correct sum of values from `rateTable` based on `startHour` and `chargeTime`

+1 value returned
- **+1/2** returns any nonconstant (derived) value
- **+1/2** returns accumulated value

Part (b)	`getChargeStartTime`	**4 points**

+1/2 invokes `getChargingCost` or replicates functionality with no errors

+1 determines charging cost
- **+1/2** considers **all** potential start times; must include at least 0 … 23
- **+1/2** determines charging cost for potential start times

Note: No penalty here for parameter passed to `getChargingCost` *that violates its preconditions (e.g., 24)*

+1 compares charging costs for two different start times

+1 determines minimum charging cost based on potential start times

Note: Penalty here for using result of call to `getChargingCost` *that violates its preconditions (e.g., 24)*

+1/2 returns start time for minimum charging cost

AP® COMPUTER SCIENCE A
2009 SCORING GUIDELINES

Question 4: Tile Game

Part (a)	`getIndexForFit`	**6 points**

+1 empty board
- **+1/2** checks for zero-sized board
- **+1/2** returns 0 if empty board detected

+1 accesses tiles from board
- **+1/2** accesses any tile from board
- **+1/2** accesses all tiles of board (as appropriate) with no out-of-bounds access potential

+1 uses tile values
- **+1/2** accesses left or right value of any tile
- **+1/2** compares left (right) value of parameter with right (left) value of any tile from board

+2 determines tile fit
- **+1/2** only right value of parameter compared with left value of initial tile of board
- **+1/2** only left value of parameter compared with right value of final tile of board
- **+1** compares appropriate values of parameter and interior tiles of board

+1 result
- **+1/2** returns located index if tile fits in board
- **+1/2** returns -1 if tile does not fit in board

Part (b)	`insertTile`	**3 points**

+1/2 invokes `getIndexForFit` or replicates functionality with no errors

+1 1/2 tile orientation
- **+1/2** invokes `rotate` on parameter
- **+1/2** performs **all** necessary rotations
- **+1/2** invokes `getIndexForFit` for each necessary orientation

+1/2 adds tile correctly and only if `getIndexForFit` returns value other than -1

+1/2 returns `true` if `getIndexForFit` returns value other than -1; `false` otherwise