AP® Computer Science AB
2008 Scoring Guidelines

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,400 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 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 excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Visit the College Board on the Web: www.collegeboard.com.
AP Central is the online home for AP teachers: apcentral.collegeboard.com.

AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 1: Anagram Set

Part A: constructor **4 points**

+1/2 `groups = new HashMap<String, HashSet<String>>();`

+1 traverse `words`
- **+1/2** correctly access an element of `words` (in context of loop)
- **+1/2** access all elements of `words` (lose this if index out-of-bounds)

+2 1/2 store anagram sets in `groups` (in context of loop)
- **+1/2** `createKeyString(`*accessedWord*`)`
- **+1** correct if keyString not already stored
- **+1** correct if keyString already stored

Part B: findLargestSets **5 points**

+1/2 construct a `HashSet<HashSet<String>>` object (must assign to a variable)

+1 1/2 iterate through anagram sets
- **+1** correctly access an anagram set (in context of loop)
- **+1/2** access all anagram sets (loop/iteration structure is correct)

+2 1/2 find largest sets
- **+1/2** determine size of anagram set
- **+1** update largest (in context of loop)
  - **+1/2** compare size with size of some previous set
  - **+1/2** add to set of sets if size >= largest so far
- **+1** correctly construct set of largest sets

+1/2 return set of largest sets

AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 2: Cache List

Part A: get **5 points**

+2 determine start location
- **+1** correct in first call to `get (front)`
- **+1** correct in subsequent calls to `get (front` or `remNode)`

+2 traverse to desired node (in context of loop)
- **+1/2** call to `getNext()`
- **+1/2** accesses more than one successive node (if needed)
- **+1** identifies desired node (may assume that `remNode` is not null)

+1/2 update `remNode` and `remIndex`

+1/2 return value at identified node

Part B: addFirst **2 1/2 points**

+1 add object
- **+1/2** create `ListNode` object containing `obj` and `front`
- **+1/2** update `front` to refer to new node

+1 1/2 update state
- **+1/2** increment `listSize`
- **+1** increment `remIndex` (if not previously -1)

Part C: Big-Oh **1 1/2 points**

+1/2 $O(n^2)$ for `LinkedList printForward` and `printReverse`
+1/2 O(n) for `APList printForward`
+1/2 $O(n^2)$ for `APList printReverse`

AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 3: MultiGrid (GridWorld)

Part A: get **2 1/2 points**

+1/2 `grid.get(loc)`

+1 1/2 null case
- **+1/2** check if `grid` contents are null
- **+1/2** construct empty set if null (nongeneric okay)
- **+1/2** return empty set if null (lose this if add set to grid)

+1/2 return `grid` contents if not null

Part B: put **2 points**

+1/2 `get(loc)` (or `grid.get(loc)` with null check)

+1/2 add `obj` to accessed set (or empty set, as required)

+1/2 `grid.put(loc,`*updatedSet*`)` (in empty case)

+1/2 correct in all cases

Part C: getNeighbors **4 1/2 points**

+1/2 construct an `ArrayList` of `Object`s (must store in a variable)

+3 1/2 add neighbors to list
- **+1/2** access a neighboring location (e.g., `grid.getNeighbors(loc)` or `loc.getAdjacentLocation(`*dir*`)`)
- **+1 1/2** traverse sets from neighboring locations
  - **+1/2** correctly access a neighboring set (in context of loop)
  - **+1** access all neighboring sets
- **+1** traverse accessed set of neighbors
  - **+1/2** correctly access an element of set (in context of loop)
  - **+1/2** access all elements of set
- **+1/2** add neighbor object to neighbor list

+1/2 return neighbor list

Note: -1 usage error for accessing `occupantMap` in Parts (A) and (B)

AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 4: Filter Objects (Design)

Part A:	OrFilter	**5 points**

+1/2 `class OrFilter implements Filter`

+1/2 declare `private` instance variable(s) capable of storing a collection of `Filter`s
Note: Two Filters suffice, as long as `add` builds and stores complex filters.

+1/2 constructor, `add`, `accept` headers correct
Note: If `add` does not return void, must return a legal value.

+1/2 constructor stores both parameters in instance variable(s) (initialize if necessary)

+1/2 `add` stores parameter in instance variable(s)

+2 1/2 `accept`
- **+1** access stored filters
  - **+1/2** correctly access a stored filter (if in a collection, must be in a loop)
  - **+1/2** access all stored filters (lose this if index out-of-bounds)
- **+1 1/2** determine whether to accept
  - **+1/2** call `accept(`*text*`)` on an accessed filter
  - **+1** return correct boolean value in all cases

Part B:	buildFilter	**4 points**

+1 access all elements of `desirable` (lose this if index out-of-bounds)

+1 construct `SimpleFilter` for each `desirable` element and `notAllowed`

+1 correctly construct all complex filters (must have at least one complex filter)

+1 build and return correct filter