AP® Computer Science AB
2005 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 4,700 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three and a half 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 official online home for the AP Program and Pre-AP: apcentral.collegeboard.com.

AP® COMPUTER SCIENCE AB
2005 SCORING GUIDELINES

2005 AB Question 1: Salmon

Part A: `nextLocation` **5 points**

+1/2 test if age < `matureAge`

+1/2 `return super.nextLocation()` if juvenile
(no credit for reimplementation)

+1 check neighboring locations
- **+1/2** correctly call `emptyNeighbors()` or `neighborsOf`
  ex: `environment().neighborsOf(<location>)` (must also call `isEmpty`)
- **+1/2** reference all (N) neighbors (must not remove the location behind)

+2 compare neighbor distance with current distance
- **+1/2** get current location (lose this if reference inaccessible field)
- **+1/2** attempt to identify closer location
- **+1** correctly compare distances from present location using `distanceHome`
  (may not (re)implement `distanceHome`, must potentially test all possibilities)

+1 return value
- **+1/2** return any empty neighbor that is closer to home
- **+1/2** return current location if none closer

Part B: `act` **4 points**

+1 distinguish juvenile from mature
- **+1/2** test if age < `matureAge`
- **+1/2** juvenile actions or mature actions, based on decision

+1/2 juvenile action: `move()`

+2 mature actions
- **+1/2** test if `location().equals(homeLocation)` (lose this if reference inaccessible field)
- **+1/2** call `breed()` if and only if at `homeLocation`
- **+1/2** call `die()` if and only if `breed()` succeeds
- **+1/2** call `move()` in context of `homeLocation` test

Note: Methods `breed()`, `die()` and `move()` may not be reimplemented, but `equals()` can be replaced by `distanceHome()` or `compareTo()` if done correctly.

+1/2 increment age (after processing)

AP® COMPUTER SCIENCE AB
2005 SCORING GUIDELINES

2005 AB Question 2: Postal Codes

Part A: efficiency **1 point**

+1/2 O(1) for `getCitiesForCode`

+1/2 O(1) for `addCityCodePair`

Part B: design **4 points**

+1 data structure
- **+1/2** supports storing city names (list or set OK)
- **+1/2** supports storing city → codes mapping

+1 initialization and declaration
- **+1/2** attempt (`new Map()` OK)
- **+1/2** correct (for data structures provided) (must be `private`)

+1 explanation
- **+1/2** describe how cities are stored
- **+1/2** describe how codes are stored and mapped

+1 `addCityCodePair` update
- **+1/2** describe how cities are updated
- **+1/2** describe how codes are updated (mapped)

Part C: `getCodesForCity` efficiency **1 point**

+1 O(log N) or better
- **+1/2** data structure supports O(log N)
- **+1/2** explanation (must identify big-Oh for structure used)

Part D: `printAllCities` **3 points**

+1 iterate and print all cities
- **+1/2** attempt to iterate/traverse and print the cities
- **+1/2** prints all, no dups
  (`System.out.println(cityToCodeMap.keySet());` is OK)

+1 cities printed in alphabetical order

+1 O(N) efficiency (must iterate over data structure that contains cities)

2005 AB Question 3: Successor Nodes

Part A: `verifyParentLinks` **5 points**

+1/2 return `true` if empty

+1/2 check parent link of root node is `null`

+1 1/2 check parent link of at least one non-root node
- **+1/2** attempt (call to `getParent` is enough)
- **+1** correct relationship test

+1 1/2 traversal
- **+1/2** attempt (recursion or iteration)
- **+1/2** traverse to bottom of tree somewhere
- **+1/2** traverse every node

+1 return correct boolean in all cases for non-empty trees

Part B: `successor` **4 points**

+1 1/2 identify and handle case where `successor` is below `t`:
- **+1/2** attempt (must test if `t.getRight() != null` and
  do something with the right subtree OR
  put in another data structure and look for successor of `t`)
- **+1** return minimum node from right subtree,
  e.g., `return minNode(t.getRight())`

+1 1/2 handle case where `successor` is above `t`:
- **+1/2** attempt traversal (loop) above `t`
  e.g., up to parent OR down from root to `t` OR
  put in another data structure and look for successor of `t`
- **+1** return `successor` node

+1 in the case where there's no successor, return `null`

Note: Must accept any correct solution, regardless of efficiency (e.g., can do inorder traversal, store nodes in a list, search for `t`, then get next entry).

Note: `equals` method not redefined in `TreeNode`, so equivalent to `==` as long as the invoking object is not `null`

2005 AB Question 4: Expand Aliases

Part A: `appendSetToQueue` **3 points**

+2 iterate over set
- **+1** attempt (must access set members)
- **+1** correct

+1 each item added to end of `q`

} no loop, no credit

Part B: `expandAlias` **6 points**

+1 instantiate set
- **+1/2** attempt (`new Set()` OK)
- **+1/2** correct

+1/2 create queue*

+1/2 enqueue `alias` or its expansion

+1 access all items in queue
- **+1/2** attempt (must access queue in body)
- **+1/2** correct

+1/2 get next alias/address (in context of loop)

+2 process alias/address (in context of loop)
- **+1/2** test whether alias or address
- **+1/2** expand alias
- **+1/2** store expansion
- **+1/2** store address in set

+1/2 return a set of addresses

*Note: Alternative data structures for queue are acceptable.

Usage: No penalty for use of "`enque`" for "`enqueue`" or "`deque`" for "`dequeue`"

2005 General Usage

Most common usage errors are 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, even if it occurs on different parts of a problem.

Non-penalized Errors

case discrepancies

variable not declared when others are declared in some part of question

missing "new" for constructor call once, when others are present in question

default constructor called without parens for example, `new Fish;`

missing { } where indentation clearly conveys intent

`obj.method` instead of `obj.method()`

loop variables used outside loop

`[r,c]`, `(r)(c)` or `(r,c)` instead of `[r][c]`

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

missing `( )` around `if/while` conditions

`length` - `size` confusion for array, `String`, and `ArrayList`, with or without `()`

missing downcast from collection or map

unnecessary construction of object whose reference is reassigned, for example

```
Direction dir = new Direction();
dir = f.Direction;
```

`private` qualifier on local variable

use "," instead of "+" for `String` in `System.out.print(str1, str2)`)

missing `;`s or missing `public`

extraneous code with no side-effect, for example a check for precondition

automatic conversion of `Integer` to `int` and vice-versa (this is legal in Java 1.5, called auto(un)boxing)

Minor Errors (1/2 point)

misspelled/ confused identifier (e.g., `len` for `length` or `left()` for `getLeft()`)

no variables declared

`new` never used for constructor calls

`void` method returns a value

modifying a constant (`final`)

use `equals` or `compareTo` method on primitives, for example
`int x; …x.equals(val)`

use value 0 for `null`

use values 0, 1 for `false`, `true`

use of `itr.next()` more than once as same value within loop

use keyword as identifier

`[]` – `get` confusion

assignment dyslexia, for example,
`x + 3 = y;` for `y = x + 3;`

`super(method())` instead of `super.method()`

formal parameter syntax (with type) in method call, e.g., `a = method(int x)`

Major Errors (1 point)

read new values for parameters or or instance variables
(prompts part of this point)

extraneous code which causes side-effect, for example, information written to output.

use interface or class name instead of variable identifier, for example `Simulation.step()` instead of `sim.step()`

`aMethod(obj)` instead of `obj.aMethod()`

use of object reference that is incorrect, for example, use of `f.move()` inside method of `Fish` class

use private data or method when not accessible

destruction of data structure (e.g. by using root reference to a `TreeNode` for traversal of the tree; this is often handled in the rubric)

use class name in place of `super` either in constructor or in method call

Note: Case discrepancies for identifiers fall under the "not penalized" category. If two identifiers differ only in capitalization, the reader may use context to differentiate between them. For example, if a student declares "Fish fish;", then uses Fish.move() instead of fish.move(), the context allows for the reader to assume the object instead of the class. If context is not clear, say if the two identifiers have the same type, then a one point penalty must be applied.

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science AB

Question 1

PART A:

```
protected Location nextLocation()
{
  if (age < matureAge)
  {
    return super.nextLocation();
  }
  else
  {
    Location currentLoc = location();
    int currentDistance = distanceHome(currentLoc);

    ArrayList possLocs = emptyNeighbors();
    for (int i = 0; i < possLocs.size(); i++)
    {
      if (distanceHome((Location)possLocs.get(i)) < currentDistance)
      {
        return (Location)possLocs.get(i);
      }
    }
    return currentLoc;
  }
}
```

PART B:

```
public void act()
{
  if ( ! isInEnv() )
  {
    return;
  }

  if (age >= matureAge && location().equals(homeLocation))
  {
    if ( breed() )
    {
      die();
    }
  }
  else
  {
    move();
  }

  age++;
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science AB

Question 2

PART A:

`getCitiesForCode:` O(1)

`addCityCodePair:` O(1)

PART B:

```
private Map cityToCodeMap;

public PostalCodeDB()
{
  . . .
  cityToCodeMap = new TreeMap();
}
```

`cityToCodeMap` will have cities for keys, and sets of codes for that city as corresponding value.

`addCityCodePair` will need to similarly update `cityToCodeMap` (adding to the set of codes for that city, adding a city entry to the map if first occurrence).

```
private Map cityToCodeMap;
private Set cities;

public PostalCodeDB()
{
  . . .
  cityToCodeMap = new HashMap();
  cities = new TreeSet();
}
```

`cityToCodeMap` will have cities for keys, and sets of codes for that city as corresponding value. `cities` will store the city names.

`addCityCodePair` will need to similarly update `cityToCodeMap` (adding to the set of codes for that city, adding a city entry to the map if first occurrence). For each new city, its name must be entered into `cities`.

PART C:

Since `cityToCodeMap` is a `TreeMap`, the `get` method is O(log N).

Since `cityToCodeMap` is a `HashMap`, the `get` method is O(1).

PART D:

```
public void printAllCities()
{
  Set cities = cityToCodeMap.keySet();
  Iterator iter = cities.iterator();
  while (iter.hasNext())
  {
    System.out.println(iter.next());
  }
}
```

OR

```
public void printAllCities()
{
  Set cities = cityToCodeMap.keySet();
  System.out.println(cities);
}
```

```
public void printAllCities()
{
  Iterator iter = cities.iterator();
  while (iter.hasNext())
  {
    System.out.println(iter.next());
  }
}
```

OR

```
public void printAllCities()
{
  System.out.println(cities);
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science AB

Question 3

PART A:

```
private boolean verifyParentLinks()
{
  return verifyParent(root, null);
}

private boolean verifyParent(TreeNode t, TreeNode parent)
{
  return (t == null ||
            (t.getParent() == parent && verifyParent(t.getLeft(), t) &&
                                        verifyParent(t.getRight(), t)));
}
```

OR

```
private boolean verifyParentLinks()
{
  if (root == null)
    return true;
  if (root.getParent()!= null)
    return false;
  return verifyChildren (root);
}

private boolean verifyChildren(TreeNode parent)
{
  if (parent == null)
    return true;
  if (parent.getLeft() != null && parent.getLeft().getParent() != parent)
    return false;
  if (parent.getRight() != null && parent.getRight().getParent() != parent)
    return false;
  return verifyChildren(parent.getLeft()) && verifyChildren(parent.getRight());
}
```

PART B:

```
private TreeNode successor(TreeNode t)
{
  if (maxNode(root) == t)
    return null;

  if (t.getRight() != null)
    return minNode(t.getRight());

  while (t.getParent() != null && t.getParent().getRight() == t)
    t = t.getParent();
  return t.getParent();
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science AB

Question 4

PART A:

```
private void appendSetToQueue(Set items, Queue q)
{
  Iterator iter = items.iterator();
  while (iter.hasNext())
  {
    q.enqueue(iter.next());
  }
}
```

PART B:

```
public Set expandAlias(String alias)
{
  Set expanded = new HashSet();

  Queue partial = new ListQueue();
  partial.enqueue(alias);
  while (!partial.empty())
  {
    String front = (String)partial.dequeue();
    if (addressBook.containsKey(front))
    {
      appendSetToQueue((Set)addressBook.get(front), partial);
    }
    else
    {
      expanded.add(front);
    }
  }
  return expanded;
}
```