AP® Computer Science AB
2005 Free-Response Questions

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.

ERRATUM NOTICE

On pages 9 and 10 of this exam, the third and fourth lines of the comments for the method `successor` should read:

```
// the method runs in time O(h), where h is the height
// of the tree rooted at root
```

COMPUTER SCIENCE AB

SECTION II

Time—1 hour and 45 minutes

Number of questions—4

Percent of total grade—50

Directions: SHOW ALL YOUR WORK, REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN Java.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Assume that the implementations of the interfaces for stacks, queues, and priority queues (pages A4-A5 in the Appendix) behave as specified.
- Assume that the implementation classes `ListNode` and `TreeNode` (page A3 in the Appendix) are used for any questions referring to linked lists or trees, unless otherwise specified.
- `ListNode` and `TreeNode` parameters may be `null`. Otherwise, unless noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.
- When Big-Oh running time is required for a response, you must use the most restrictive Big-Oh expression. For example, if the running time is $O(n)$, a response of $O(n^2)$ will not be given credit.

1. This question involves reasoning about the code from the Marine Biology Simulation case study. A copy of the code is provided as part of this exam.

   Consider defining a new type of fish called `Salmon` that keeps track of its age and has different behaviors based on its age. A salmon has two phases in its life: juvenile and mature. The age at which a salmon changes from juvenile to mature is called the *age of maturity*. The age of a salmon is the number of times its `act` method executes — the number of time-steps in a simulation. When the age is less than the age of maturity, a salmon is in the juvenile phase. When the age is greater than or equal to the age of maturity, a salmon is in the mature phase. A `Salmon` should age one time-step as the last statement executed in its `act` method.

   The behavior for `Salmon` is described as follows.

   - A juvenile `Salmon` moves like a `Fish` but does not `breed` or `die`.
   - A mature `Salmon` does not die until it successfully breeds. A mature `Salmon` cannot breed until it returns to the location where it was constructed. We call this location its home. When a `Salmon` reaches maturity, it attempts to move back to its home. A mature `Salmon` moves only if it can move to a neighboring location whose distance is closer to home than its current distance to home. Once the mature `Salmon` gets home, it stays in that home location until it breeds. A mature `Salmon` that starts a simulation step in its home location attempts to breed like a normal `Fish`. If the breed attempt is successful, the `Salmon` dies in the same turn.

The partial declaration for the `Salmon` class is shown below. You will override the `nextLocation` and `act` methods.

```
public class Salmon extends Fish
{

  private Location homeLocation;
    // location where created

  private int age;
    // number of times act has been called on this Salmon

  private int matureAge;
    // age at which a Salmon becomes mature


  // constructs a Salmon at the specified location in a given
  // environment; age is initialized to 0, homeLocation is set to loc,
  // and matureAge is set to maturityAge
  // precondition:  parameters are non-null, loc is valid for env
  public Salmon(Environment env, Location loc, int maturityAge)
  {
    super(env, loc);
    age = 0;
    matureAge = maturityAge;
    homeLocation = loc;
  }

  // other constructors not shown


  // returns the number of moves required to move
  // from loc to the fish's homeLocation
  private int distanceHome(Location loc)
  {  /* implementation not shown */  }


  // a juvenile Salmon chooses its next location as if it were
  // an ordinary fish;
  // a mature Salmon chooses an empty neighboring location that is
  // closer to home than this fish's current location, if one exists;
  // otherwise, it returns this fish's current location
  // postcondition: the state of this Salmon is unchanged
  protected Location nextLocation()
  {  /* to be implemented in part (a) */  }


  // acts for one step of the simulation
  public void act()
  {  /* to be implemented in part (b) */  }

  // implementation of generateChild not shown
}
```

(a) Override the `nextLocation` method for the `Salmon` class. A juvenile `Salmon` chooses its `nextLocation` as if it were an ordinary `Fish`. A mature `Salmon` chooses its `nextLocation` by finding any empty neighboring location with a smaller distance to home than its current location. If no such location exists, the mature `Salmon` stays in its current location.

In writing `nextLocation`, you may use any of the accessible methods of the classes in the case study, including those specified in this question.

Complete method `nextLocation` below.

```
// a juvenile Salmon chooses its next location as if it were
// an ordinary fish;
// a mature Salmon chooses an empty neighboring location that is
// closer to home than this fish's current location, if one exists;
// otherwise, it returns this fish's current location
// postcondition: the state of this Salmon is unchanged
protected Location nextLocation()
```

(b) Override the `act` method for the `Salmon` class. A juvenile `Salmon` does not breed or die but will move as an ordinary `Fish`. If a mature `Salmon` is not in its home location, it will attempt to move; otherwise, it attempts to breed. A mature `Salmon` does not die until it successfully breeds. A `Salmon` ages one time-step as the last statement executed in the `act` method.

In writing `act`, you may call any of the accessible methods of the classes in the case study, including those specified in this question.

Complete method `act` below.

```
// acts for one step of the simulation
public void act()
```

2. Consider the design of a class to store cities and postal codes.

A city, such as Miami, Florida, may have several postal codes, such as 33199 and 33174. A postal code may occasionally represent more than one city; for example, the postal code 95324 is used for both Hilmar, California, and Irwin, California. Postal codes may include letters as well as digits, such as postal code H3C5T7 used for Montreal, Quebec.

The class `PostalCodeDB` stores information about cities and postal codes. The initial version of the class is shown below. Cities and postal codes are represented as strings in this class.

```
public class PostalCodeDB
{
  private Map codeToCityMap; // each key is a postal code, its
                             // associated value is a set of cities

  // constructs an empty database
  public PostalCodeDB()
  {
    codeToCityMap = new HashMap();
  }


  // returns the set of cities associated with postalCode if that
  // code is in the database, otherwise returns null
  public Set getCitiesForCode(String code)
  {
    return (Set) codeToCityMap.get(code);
  }


  // adds a new city, postalCode pair to this database
  public void addCityCodePair(String code, String city)
  {
    Set citySet = getCitiesForCode(code);
    if (citySet == null)
    {
      citySet = new HashSet();
      codeToCityMap.put(code, citySet);
    }
    citySet.add(city);
  }

}
```

(a) For the implementation shown above, give the expected Big-Oh running time for the `PostalCodeDB` methods `getCitiesForCode` and `addCityCodePair`. Express your answers in terms of *N*, where *N* is the number of city-code pairs that have been added to the `PostalCodeDB`.

Method	Expected Big-Oh Running Time
`getCitiesForCode`	
`addCityCodePair`	

(b) The `PostalCodeDB` class is to be modified to provide two additional operations.

1. A new method, `getCodesForCity`, should return a set of all postal codes for a city. This method must run in time $O(\log N)$ or better, where N is the number of pairs that have been added to this `PostalCodeDB`.
2. A new method, `printAllCities`, should print an alphabetical list of all cities. This method must run in time $O(N)$ or better.

In the spaces provided below, describe your modifications to the design of the `PostalCodeDB` class to support the new functionality listed above.

- Declare any additional instance variables that are required in the `PostalCodeDB` class. Show the initialization of these variables in the `PostalCodeDB` constructor.

```
// additional instance variable(s)




// constructs an empty database
public PostalCodeDB()
{
  codeToCityMap = new HashMap();

  // initialization of additional instance variable(s)




}
```

- Explain how the `PostalCodeDB` class will use these variables to organize the data.

- Describe any changes that are needed in the method `addCityCodePair` to update these variables.

(c) Justify how `PostalCodeDB` method `getCodesForCity` can meet the required running time of $O(\log N)$ or better.

(d) Write the `PostalCodeDB` method `printAllCities`, which prints an alphabetical list of all the cities in the `PostalCodeDB`. This operation must run in time $O(N)$ or better, where N is the number of city-code pairs that have been added to the `PostalCodeDB`.

Complete method `printAllCities` below.

```
// prints an alphabetical list of all cities in this database
// postcondition: the state of the data structure is not changed
public void printAllCities()
```

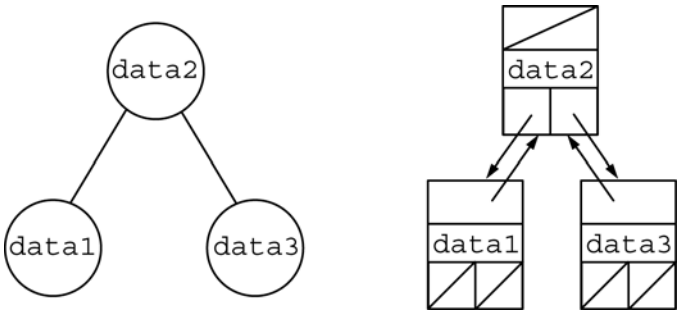
3. In this question, assume that the `TreeNode` class (whose code is provided in the Appendix) has been modified as shown below to include an additional link to the node's parent. The following private instance variable and two public methods have been added to the `TreeNode` class to provide this link.

```
private TreeNode parent;

public TreeNode getParent()
{  return parent;  }

public void setParent(TreeNode p)
{  parent = p;  }
```

In the following diagrams, the three-node tree on the left is expanded on the right, showing how parent and child nodes are linked.



Consider the following `Tree` class that maintains and processes a binary search tree whose nodes are objects of the modified `TreeNode` class. You may assume that the tree contains `Comparable` objects and that no two objects stored in the tree are equal. You will implement two of the private methods of the `Tree` class.

```
public class Tree
{
  private TreeNode root;

  // constructs a new, empty tree
  public Tree()
  {  root = null;  }


  // returns the node containing the smallest value in the tree
  // rooted at t when t is not null; otherwise, returns null;
  // the method runs in time O(h), where h is the height
  // of the tree rooted at t
  private TreeNode minNode(TreeNode t)
  {  /* implementation not shown */  }


  // returns the node containing the largest value in the tree
  // rooted at t when t is not null; otherwise, returns null;
  // the method runs in time O(h), where h is the height
  // of the tree rooted at t
  private TreeNode maxNode(TreeNode t)
  {  /* implementation not shown */  }


  // returns true if all nodes in this tree have correct parent links;
  // otherwise, returns false;
  // each node should be the parent of its left and right children;
  // the root node's parent should be null
  private boolean verifyParentLinks()
  {  /* to be implemented in part (a) */  }


  // returns the node which is the successor of t in this tree;
  // returns null if t contains the maximum value in this tree;
  // the method runs in time O(h), where h is the height
  // of the tree rooted at t
  // precondition:  t is a node in this tree;
  //                all nodes in this tree have correct parent links
  private TreeNode successor(TreeNode t)
  {  /* to be implemented in part (b) */  }

}
```

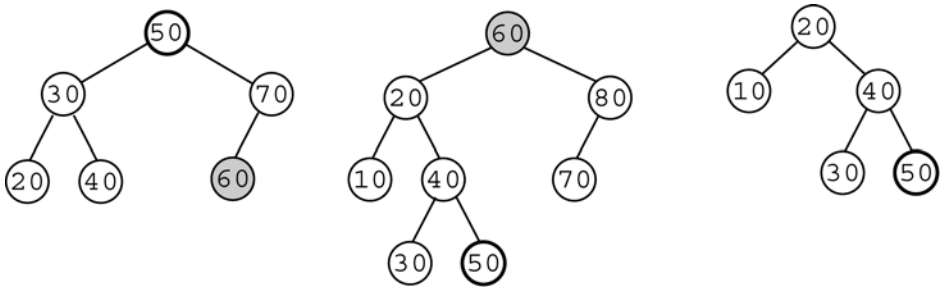
(a) Write the `Tree` method `verifyParentLinks`, which tests the structure of the tree. The method traverses the tree and checks whether the parent link in each node correctly references that node's parent. The parent link for the root node should be `null`.

In writing `verifyParentLinks`, you may write and use a helper method.

Complete method `verifyParentLinks` below.

```
// returns true if all nodes in this tree have correct parent links;
// otherwise, returns false;
// each node should be the parent of its left and right children;
// the root node's parent should be null
private boolean verifyParentLinks()
```

(b) Write the `Tree` method `successor`. The successor of a given tree node is defined as the next node that would be visited after the given node during an in-order traversal of the tree. For example, in the first two trees drawn below, the shaded node is the successor of the node containing 50. In the third tree, the node containing 50 is the largest node in the tree and has no successor.



In writing `successor`, you may use any of the accessible methods in the `Tree` and modified `TreeNode` classes.

Complete method `successor` below.

```
// returns the node which is the successor of t in this tree;
// returns null if t contains the maximum value in this tree;
// the method runs in time O(h), where h is the height
// of the tree rooted at t
// precondition:  t is a node in this tree;
//                all nodes in this tree have correct parent links
private TreeNode successor(TreeNode t)
```

4. Consider an e-mail address book that allows the user to store e-mail addresses by associating them with an alias (a name used to identify the recipient or group of recipients). The address book is represented by a map in which the key is the alias and the corresponding value is a set of the actual e-mail addresses and/or other aliases associated with that key. Each alias is a key in the map representing the address book; e-mail addresses are not keys in the map. This distinguishes aliases from addresses.

   The following shows an example of an address book.

   addressBook

Alias	Expansion Set
techstaff	{ pat@ez.edu, chris@ez.edu }
faculty	{ bobby, ana, sam@ez.edu }
bobby	{ bob@cs.org }
ana	{ ana@ez.edu }
all	{ phil@ez.edu, faculty, techstaff }

   The e-mail address book is represented by the class `EmailAddresses`, for which a partial declaration is shown below. You will implement two methods in the `EmailAddresses` class.

```
public class EmailAddresses
{
  private Map addressBook;
    // Map addressBook is structured such that each key is a String
    // representing an alias and the corresponding value is a Set
    // containing Strings that represent e-mail addresses and aliases

  // constructors, other data fields and other methods not shown


  // adds all elements of the Set items to the Queue q
  // postcondition: the Set items is not modified;
  //                q contains its original elements followed
  //                by the elements from items
  private void appendSetToQueue(Set items, Queue q)
  {  /* to be implemented in part (a) */  }


  // returns a Set consisting of the e-mail addresses associated with
  // the parameter alias, such that all intermediate resulting aliases
  // have been expanded;
  // precondition: alias is a key in addressBook
  public Set expandAlias(String alias)
  {  /* to be implemented in part (b) */  }

}
```

(a) Write the `EmailAddresses` method `appendSetToQueue`. This method adds the elements of the set parameter `items` to the queue parameter `q`. The table below shows the result of several calls to `appendSetToQueue`. Note that the front of each queue is to the left and the rear of each queue is to the right.

Set `items` (unchanged after each call)	`q` Before Call to `appendSetToQueue`	`q` After the Call `appendSetToQueue(items, q)`
`{ pat@ez.edu, chris }`	`[]`	`[pat@ez.edu, chris]`
`{ bob@cs.org }`	`[pat, ana]`	`[pat, ana, bob@cs.org]`
`{ sam@ez.edu, bobby }`	`[chris]`	`[chris, sam@ez.edu, bobby]`

Complete method `appendSetToQueue` below.

```
// adds all elements of the Set items to the Queue q
// postcondition: the Set items is not modified;
//                q contains its original elements followed
//                by the elements from items
private void appendSetToQueue(Set items, Queue q)
```

(b) In this part, you will write the `EmailAddresses` method `expandAlias`. When an alias is expanded, the expansion can contain aliases that may also need to be expanded. The sequence of expanding all such aliases stops when no further expansion is possible; that is, the original alias has been resolved into a set of e-mail addresses.

Note that when an alias is expanded, the sequence of expansions could result in a cycle in which the original alias is expanded again. An example of a cycle occurs when expanding either of the aliases shown in the following table. For this question, you may assume that an alias cycle never occurs.

Alias	Expansion Set
`players`	`{ offense, defense }`
`offense`	`{ players }`

One strategy for fully expanding an alias is to use an intermediate queue to hold aliases that need to be expanded and a set to hold the resulting e-mail addresses. When an alias is removed from the queue, its associated expansion is added to the queue. If an e-mail address is removed from the queue, it is added to the result set. This process is repeated until the queue is empty.

For example, the following steps would be taken during the expansion of the alias `all` from the `Map addressBook` shown in the example on the next page. The first step is to add the alias `all` to the queue. The resulting set of e-mail addresses is initially empty as shown below.

(Front) (Rear)

Queue `[all]`

Set `{ }`

The first item is removed from the queue. Because it is an alias (`all`), the members of its expansion set are added to the queue as shown in the following table.

(Front) (Rear)

Queue `[phil@ez.edu, faculty, techstaff]`

Set `{ }`

The next item that is removed from the queue is an e-mail address (`phil@ez.edu`), which is added to the result set of e-mail addresses associated with the original alias. After this step the queue and set contents are as shown.

	(Front)	(Rear)
Queue	`[faculty, techstaff]`	
Set	`{ phil@ez.edu }`	

This process continues until the queue is empty. The set of e-mail addresses resulting from the expansion of the alias `all` is shown below.

```
{ phil@ez.edu, sam@ez.edu, pat@ez.edu, chris@ez.edu,
  bob@cs.org, ana@ez.edu }
```

Write the `EmailAddresses` method `expandAlias`. This method returns a set containing all the e-mail addresses that are associated with the parameter `alias`.

The table below shows the map given at the beginning of the question, repeated here for your convenience.

`addressBook`

Alias	Expansion Set
`techstaff`	`{ pat@ez.edu, chris@ez.edu }`
`faculty`	`{ bobby, ana, sam@ez.edu }`
`bobby`	`{ bob@cs.org }`
`ana`	`{ ana@ez.edu }`
`all`	`{ phil@ez.edu, faculty, techstaff }`

The next table shows the result of several calls to `expandAlias`. (The method returns a set; it is acceptable if your implementation produces the elements in a different order.)

Method Call	Set Returned from Method Call
`expandAlias("bobby")`	`{ bob@cs.org }`
`expandAlias("faculty")`	`{ sam@ez.edu, bob@cs.org, ana@ez.edu }`
`expandAlias("all")`	`{ phil@ez.edu, sam@ez.edu, pat@ez.edu, chris@ez.edu, bob@cs.org, ana@ez.edu }`

In writing `expandAlias`, you may use the method `appendSetToQueue` defined for the `EmailAddresses` class specified for this problem. Assume that this method works as specified, regardless of what you wrote in part (a).

You may assume that a class `ListQueue` implementing the `Queue` interface has been provided.

Complete method `expandAlias` below.

```
// returns a Set consisting of the e-mail addresses associated with
// the parameter alias, such that all intermediate resulting aliases
// have been expanded;
// precondition: alias is a key in addressBook
public Set expandAlias(String alias)
```

END OF EXAM