Problem 1—Short answer (10 points)
1a) Trace through the evaluation of the following program and indicate the output it produces.

```java
import acm.program.*;

public class Mystery extends ConsoleProgram {
    public void run() {
        int[] array = new int[13];
        for (int i = 1; i <= 12; i++) {
            for (int j = i; j > 0; j--) {
                array[j] += j;
            }
        }
    }
}
```

Work through the method carefully and indicate your answer by filling in the boxes below to show the final contents of `array`, just before the `run` method returns:

<table>
<thead>
<tr>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
1b) Suppose that you have been assigned to take over a project from another programmer who has just been dismissed for writing buggy code. One of the methods you have been asked to rewrite—which doesn’t even compile—looks like this:

```java
/**
 * Returns the acronym of the specified string, which is defined to be a new word formed from the first letter in each separate word contained in the string. A word is a consecutive string of letters. For example, calling
 *    acronym("self-contained underwater breathing apparatus")
 * should return the string "scuba".
 */
private String acronym(String str) {
    boolean inWord = false;
    for (int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        if (ch.isLetter()) {
            if (!inWord) result += ch;
            if (!inWord) result += ch;
        } else {
            inWord = false;
        }
    }
    return result;
}
```

Unfortunately, the code—short as it is—contains several bugs. Circle the bugs in the implementation and write a short sentence explaining the precise nature of each problem you identify.

Problem 2—Using the acm.graphics library (15 points)

Back before we had a graphics library, one of the early assignments in CS 106A was to write a program that played a simple game called Nim. In the simplest version of the game, two players start with a pile of 11 coins on the table between them. The players then take turns removing 1, 2, or 3 coins from the pile. The player who is forced to take the last coin loses. In the old days, the point of this assignment was to figure out a strategy for winning the game. With our modern attachment to fancier user interfaces, however, it is just as easy to focus on the problem of representing the game graphically on the screen.

Your job in this problem is to complete the implementation of the program that appears on the following page, which is responsible for displaying the coins and handling the user interaction. You should think about the task as consisting of two parts. The first part of this problem is to create a graphical display in which you have a line of coins arranged horizontally on the screen, like this:
In constructing your display, you should make use of the named constants—\texttt{N\_COINS}, \texttt{COIN\_SIZE}, and \texttt{COIN\_SEP}—that appear at the top of the program framework. You should also make sure that each coin is filled in \texttt{GRAY} and outlined in \texttt{BLACK}. The line of coins should be centered both horizontally and vertically in the window.

The second part of the problem consists of making it possible to take coins away. Add the necessary code so that if the user clicks the mouse in one of the last three coins in the row, those coins disappear. For example, if the user clicked on the third coin from the right end, the program should respond by removing the last three coins from the display, like this:

If the user then clicked on the rightmost coin, only that coin should go away:

If the mouse click does not occur inside a coin or if the coin is not one of the last three in the row, that click should simply be ignored.

In order to solve this problem, you will need to store the \texttt{GOval} objects in some kind of data structure that allows you to keep track of their order. Given the fact that you need to take coins away, a \texttt{List} seems perfect for the job. When a mouse click occurs, your program needs to find the object at that mouse location (if any) and see whether it is one of the last three elements in the list. If so, your program should remove that element and any following elements from both the list and the window.

You may assume the definition of the following constants:

\begin{verbatim}
public static final int N_COINS = 11;    /* Number of coins */
public static final int COIN_SIZE = 25;   /* Diameter of a coin */
public static final int COIN_SEP = 10;    /* Space between coins */
\end{verbatim}
Problem 3—Strings (15 points)

In Dan Brown’s best-selling novel *The Da Vinci Code*, the first clue in a long chain of puzzles is a cryptic message left by the dying curator of the Louvre. Two of the lines of that message are

\[ O, Draconian devil! \]
\[ Oh, lame saint! \]

Professor Robert Langdon (the hero of the book, played by Tom Hanks in the movie) soon recognizes that these lines are **anagrams**—pairs of strings that contain exactly the same letters even if those letters are rearranged—for

\[ Leonardo da Vinci \]
\[ The Mona Lisa \]

Your job in this problem is to write a predicate method

```java
public boolean isAnagram(String s1, String s2)
```

that takes two strings and returns `true` if they contain exactly the same alphabetic characters, even though those characters may appear in any order. Thus, your method should return `true` for each of the following calls:

- `isAnagram("O, Draconian devil!", "Leonardo da Vinci")`
- `isAnagram("Oh, lame saint!", "The Mona Lisa")`
- `isAnagram("ALGORITHMICALLY", "logarithmically")`
- `isAnagram("Doctor Who", "Torchwood")`

These examples illustrate two important requirements for the `isAnagram` method:

- The implementation should look only at letters (i.e., characters for which the `Character.isLetter` method returns `true`), ignoring any extraneous spaces and punctuation marks thrown in along the way.
- The implementation should ignore the case of the letters in both strings.

There are many different algorithmic strategies you could use to decide whether two strings contain the same alphabetic characters. If you’re having trouble coming up with a strategy, you should note that two strings are anagrams if and only if they would generate the same letter-frequency table, as discussed in the lecture on arrays.
Problem 4—Arrays (10 points)

So the last shall be first.
—King James Bible, Matthew 20:16

Write a function `rotateRight` that takes an array of `doubles` and that has the effect of shifting every element one position rightward toward the end of the array, with the exception of the last element, which gets moved to the first position. For example, if the array `values` has the contents

<table>
<thead>
<tr>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14</td>
</tr>
<tr>
<td>2.72</td>
</tr>
<tr>
<td>98.6</td>
</tr>
<tr>
<td>6.28</td>
</tr>
<tr>
<td>22.5</td>
</tr>
<tr>
<td>42.0</td>
</tr>
</tbody>
</table>

calling

`rotateRight(values);`

should shift the first five values one position to the right and move the value from the last position to the front, as follows:

<table>
<thead>
<tr>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.0</td>
</tr>
<tr>
<td>3.14</td>
</tr>
<tr>
<td>2.72</td>
</tr>
<tr>
<td>98.6</td>
</tr>
<tr>
<td>6.28</td>
</tr>
<tr>
<td>22.5</td>
</tr>
</tbody>
</table>
Problem 5—Building graphical user interfaces (20 points)

Many web sites today—including, for example, reddit.com and digg.com—allow users to vote on entries, which in turn affects the rating of items displayed on that site. Suppose that you want to do the same for a site whose purpose is to display pictures and let users raise the score associate with each pictures by clicking on it. Since this site is something of a Digg for pictures, we’ll call it Pigg.

The names of the image files that you need to display are stored in a data file named Piggs.txt, where the filenames appear one per line. For test purposes, we’ll use the following file, which includes images of several famous pigs:

```
Babe.png
KingPig.png
MissPiggy.png
Piglet.png
Wilbur.png
```

You may assume that each of these files exists in your images directory and that each one corresponds to an image whose size is given by the following constants:

```
private static final int IMAGE_WIDTH = 100;
private static final int IMAGE_HEIGHT = 100;
```

The first thing you need to do is read this data file and display the images at the top of the graphics window, moving from left to right across the window. Centered underneath each image, you should create a GLabel initialized to show the value 0 in a 16-point sans-serif font. After the conclusion of this step, the window should look like this:

The interactive part of this problem is that you need to be able to click on one of these images and have your program add one to the score shown in the associated GLabel. For example, if you click on King Pig from Angry Birds, the window will look like this:
In solving this problem, you should keep the following points in mind:

- You don’t need to worry about what happens if the number of images listen in the file is too large to fit in the window. You should simply position the first image at the upper left corner and then position each subsequent image immediately to the right of the last one.

- The centering of the label must occur whenever its value changes. If the count displayed underneath one of the images goes from 9 to 10, the starting point of the label will have to move leftward to keep the image centered.
Problem 6—Data structures (20 points)

In recent years, the globalization of the world economy has put increasing pressure on software developers to make their programs operate in a wide variety of languages. That process used to be called internationalization, but is now more often referred to (perhaps somewhat paradoxically) as localization. In particular, the menus and buttons that you use in a program should appear in a language that the user knows.

Your task in this problem is to write a definition for a class called Localizer designed to help with the localization process. The constructor for the class has the form

```java
public Localizer(String filename)
```

The constructor creates a new Localizer object and initializes it by reading the contents of the data file. The data file consists of an English word, followed by any number of lines of the form

```
x=x=translation
```

where `x` is a standardized two-letter language code, such as `de` for German, `es` for Spanish, and `fr` for French. Part of such a data file, therefore, might look like this:

```
Localizations.txt
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

This file tells us, for example, that the English word Cancel should be rendered in German as Abbrechen, in Spanish as Ayudar, and in French as Annuler.

Beyond the constructor, the only public method you need to define for Localizer is

```java
public String localize(String word, String language)
```

which returns the English word passed as the first parameter translated into the language indicated by the two-letter language code passed as the second parameter. For example, assuming that you had initialized a variable `myLocalizer` by calling

```java
Localizer myLocalizer = new Localizer("Localizations.txt");
```

you could then call
myLocalizer.localize("Open", "de");

and expect it to return the string "Öffnen". If no entry appears in the table for a particular word, localize should return the English word unchanged. Thus, OK becomes Approuver in French, but would remain as OK in Spanish or German.

As you write your answer to this problem, here are a few points to keep in mind:

• You can determine when a new entry starts in the data file by checking for a line without an equal sign. As long as an equal sign appears, what you have is a new translation for the most recent English word into a new language.

• For this problem, you don’t have to work about distinctions between uppercase and lowercase letters and may assume that the word passed to localize appears exactly as it does in the data file.

• The data file shown on the previous page is just a small example; your program must be general enough to work with a much larger file. You may not assume that there are only three languages or, worse yet, only four words.

• You don’t have to do anything special for the characters in other languages that are not part of standard English, such as the ö and ß that appear in this data file. They’re just characters in the expanded Unicode set that Java uses.

• It may help you solve this problem if you observe that it is the combination of an English word and a language code that has a unique translation. Thus, although there are several different translations of the word Close in the localizer and German translations of many words, there is only one entry for the combination Close+de.
Problem 7—Essay: Extensions to the assignments (10 points)

In many ways, the assignment that offers the greatest opportunities for extension is the Breakout game from Assignment #3. For this problem, imagine that you are trying to create a “Super Breakout” in which multiple balls can be in play at the same time.

Here is how you would like your new program to work. In the initial layout of bricks at the top of the screen, a random collection of bricks (a different set each time) are designated superbricks and are marked on the display by having a black border instead of being solidly colored throughout. Superbricks are relatively rare; as you lay them out, each brick has a 5% chance of being a superbrick. One possible initial configuration with four superbricks appears at the left side of Figure 1 at the bottom of the page.

The game proceeds exactly as it did before until the ball hits a superbrick, at which point the program fires off another ball just as it did at the beginning of the turn. The diagram at the right of Figure 1 shows the situation shortly after the ball has collided with the cyan superbrick in the second row. There are now two balls in play. In each time step of the game, both balls move, bounce off the walls and the paddle, and collide with bricks (and possibly each other). The number of balls increases as you hit more superbricks, and decreases when a ball falls off the bottom. The current turn, however, ends only when there are no balls remaining on the board.

Discuss the changes that would be necessary in the Breakout implementation to implement this extension to the game. Keep in mind that this is an essay question, and you should not feel compelled to write any actual code unless you feel that doing so is the best way to convey your ideas. You should, however, explain as clearly as you can what parts of the program will need to change and what data structures you would need to add to the program to implement this feature.

Figure 1. Scenes from Super Breakout