Part A: decodeString 4 1/2 points

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

+2 retrieve substrings from masterString
+1/2 correctly call getStart() and getLength() on accessed part
+1 1/2 extract a substring from masterString
   +1/2 masterString.substring(X,Y)
   +1 extract correct substring

+1 1/2 build and return decoded string
  +1 correctly build string from substrings of masterString
  +1/2 return built string

Part B: encodeString 4 1/2 points

+1/2 construct an ArrayList<StringPart> (must assign to a variable, generic okay)

+3 1/2 find, collect string parts, and build list (in context of loop)
  +1 findPart(X), where X is word or a substring of word
  +1 calls to findPart involve progressively smaller suffixes of word
  +1/2 add found string part to ArrayList of string parts
  +1 build correct list of string parts (must have used findPart)

+1/2 return ArrayList of string parts
Question 2: String Coder

PART A:

public String decodeString(ArrayList<StringPart> parts) {
    String expanded = "";
    for (StringPart nextPart : parts) {
        int ending = nextPart.getStart() + nextPart.getLength();
        expanded += masterString.substring(nextPart.getStart(), ending);
    }
    return expanded;
}

PART B:

public ArrayList<StringPart> encodeString(String word) {
    ArrayList<StringPart> parts = new ArrayList<StringPart>();
    while (word.length() > 0) {
        StringPart nextPart = findPart(word);
        parts.add(nextPart);
        word = word.substring(nextPart.getLength());
    }
    return parts;
}

ALTERNATE SOLUTION:

public ArrayList<StringPart> encodeString(String word) {
    ArrayList<StringPart> parts = new ArrayList<StringPart>();
    int index = 0;
    while (index < word.length()) {
        StringPart nextPart = findPart(word.substring(index));
        parts.add(nextPart);
        index += nextPart.getLength();
    }
    return parts;
}
(a) Write the `StringCoder` method `decodeString`. This method retrieves the substrings in the master string represented by each of the `StringPart` objects in `parts`, concatenates them in the order in which they appear in `parts`, and returns the result.

Complete method `decodeString` below.

```java
/** @param parts an `ArrayList` of string parts that are valid in the master string
 * @Precondition: parts.size() > 0
 * @return the string obtained by concatenating the parts of the master string
 */
public String decodeString(ArrayList<StringPart> parts) {
    String result = "";
    for (StringPart s : parts) {
        int start = s.getStart();
        int length = s.getLength();
        String temp = masterString.substring(start, start + length);
        result += temp;
    }
    return result;
}
```
(b) Write the `StringCoder` method `encodeString`. A string is encoded by determining the substrings in the master string that can be combined to generate the given string. The encoding starts with a string part that matches the beginning of the word, followed by a string part that matches the beginning of the rest of the word, and so on. The string parts are returned in an array list in the order in which they appear in `word`.

The helper method `findPart` must be used to choose matching string parts in the master string.

Complete method `encodeString` below.

```java
/**
 * @param word the string to be encoded
 * @Precondition: all of the characters in word appear in the master string;
 * @return an ArrayList of string parts of the master string that can be combined
 * to create word
 */
public ArrayList<StringPart> encodeString(String word) {
    ArrayList<StringPart> list = new ArrayList();
    String str = word;
    while (str.length() > 0) {
        StringPart s = findPart(str);
        list.add(s);
        int len = s.getLength();
        if (len <= str.length())
            str = str.substring(len);
    }
    return list;
}
```
(a) Write the `StringCoder` method `decodeString`. This method retrieves the substrings in the master string represented by each of the `StringPart` objects in `parts`, concatenates them in the order in which they appear in `parts`, and returns the result.

Complete method `decodeString` below.

```java
/** @param parts an ArrayList of string parts that are valid in the master string
 * Precondition: parts.size() > 0
 * @return the string obtained by concatenating the parts of the master string */
public String decodeString(ArrayList<StringPart> parts)
{
    String result = new String();
    for (int i = 0; i < parts.size(); i++)
    {
        int start = parts.get(i).getStart();
        result += masterString.substring(start, start + parts.get(i).getLength());
    }
    return result;
}
```

GO ON TO THE NEXT PAGE.
(b) Write the StringCoder method encodeString. A string is encoded by determining the substrings in the master string that can be combined to generate the given string. The encoding starts with a string part that matches the beginning of the word, followed by a string part that matches the beginning of the rest of the word, and so on. The string parts are returned in an array list in the order in which they appear in word.

The helper method findPart must be used to choose matching string parts in the master string.

Complete method encodeString below.

```java
/** @param word the string to be encoded
 *   @Precondition: all of the characters in word appear in the master string;
 *   @return an ArrayList of string parts of the master string that can be combined
 *           to create word
 */
public ArrayList<StringPart> encodeString(String word) {
    String temp = word;
    ArrayList<StringPart> parts;
    for (int i = 0; i < word.length(); i++) {
        parts.add(findPart(word));
        temp = temp.substring(0);
```
(a) Write the StringCoder method decodeString. This method retrieves the substrings in the master
string represented by each of the StringPart objects in parts, concatenates them in the order in
which they appear in parts, and returns the result.

Complete method decodeString below.

```java
/**
 * @param parts an ArrayList of string parts that are valid in the master string
 * @Precondition: parts.size() > 0
 * @return the string obtained by concatenating the parts of the master string
 */
public String decodeString(ArrayList<StringPart> parts)
{
    String word = "";
    for (int i = 0; i < parts.size(); i++)
    {
        word += substring(parts.getStart(), parts.getLength() + 1);
    }
    return word;
}
```
(b) Write the `StringCoder` method `encodeString`. A string is encoded by determining the substrings in the master string that can be combined to generate the given string. The encoding starts with a string part that matches the beginning of the word, followed by a string part that matches the beginning of the rest of the word, and so on. The string parts are returned in an array list in the order in which they appear in `word`.

The helper method `findPart` must be used to choose matching string parts in the master string.

Complete method `encodeString` below.

```java
/**
 * @param word the string to be encoded
 *  *    Precondition: all of the characters in word appear in the master string;
 *  *    word.length() > 0
 *  * @return an ArrayList of string parts of the master string that can be combined
 *        to create word
 */
public ArrayList<StringPart> encodeString(String word)
{
    ArrayList<StringPart> parts = new ArrayList<StringPart>;
    for (int key = 0; key < master.length(); key++)
    {
        int length = 1;
        while (length > 0)
        {
            if (word.substring(w, length).equals(master.substring(key, length)))
                length += ;
            else
                if (length > 1)
                    parts.add(new StringPart(key, length));
                length = 0;
        }
    }
    return parts;
}
```

GO ON TO THE NEXT PAGE.
AP® COMPUTER SCIENCE A
2008 SCORING COMMENTARY

Question 2

Overview

This question focused on abstraction, string manipulation, ArrayList traversal, and algorithm implementation. Students were provided the framework of a StringPart class for identifying a substring of a string (by specifying start index and length). Using this idea of a string part, algorithms for encoding and decoding strings as sequences of parts of a master string were described. In part (a) students were required to implement the described algorithm for decoding a string given its representation as an ArrayList of StringPart objects. This involved traversing the ArrayList, accessing the appropriate substrings in the master string (using the substring method), and concatenating the substrings to obtain the original string. In part (b) students had to implement the encoding algorithm, which involved constructing an ArrayList of StringPart objects that represented the given string. A helper method, findPart, was provided for extracting the individual string parts, which had to be added to an ArrayList in sequence.

Sample: A2A
Score: 9

In part (a) the solution utilizes a for-each loop to access all the elements of ArrayList<StringPart> parts. Outside the loop, the student declares and initializes a String identified as result. Auxiliary variables start and length are declared inside the loop body and assigned correct values based on the results returned by the getStart and getLength methods from the StringPart class called on an element of parts. The correct start and start+length parameters are used in the masterString.substring call and the resulting value is stored in String temp. The += operator is then used to concatenate temp with result, which is returned after the loop exit.

In part (b) ArrayList<StringPart> list is declared and instantiated; the fact that <StringPart> is not used with the constructor does not affect correctness. A local String variable str is assigned the value of the method's parameter word and then str is used throughout the method. The context of a loop is established by while (str.length()>0). The call to findPart is syntactically correct and the result is stored in StringPart s, which is then added to the ArrayList list. Then int len is declared and assigned the result from s.getLength(). Excessive code that causes no side effect, such as the meaningless guard of the update of str since the condition will always be true, is not penalized. The call of the String method substring str = str.substring(len) gives the correct value of str to be used as the parameter in the next call to findPart. The correctly built ArrayList list is returned after the loop exit.

Sample: A2B
Score: 6

In part (a) the solution utilizes a traditional for loop with int i as the loop control variable and an ending condition of i <parts.size(). Outside the loop, the student declares result and uses a String constructor to initialize to an empty string. Within the loop, each element of parts is accessed by using parts.get(i). This is correct, but much less code would have been required if a for-each loop had been used. Duplicate calls are made to parts.get(i) but int start is assigned the result returned by the call to parts.get(i).getStart() to avoid a duplicate call of getStart. The correct start and start+parts.get(i).getLength() parameters are used in the masterString.substring call, and the resulting value is immediately concatenated with result. The return result statement then appears after the loop exit.

© 2008 The College Board. All rights reserved.
Visit the College Board on the Web: www.collegeboard.com.
In part (b) ArrayList<StringPart> parts is declared but not instantiated so the ½ point for constructing an ArrayList of StringParts was not awarded. The context of a loop is established, even though the loop is never finished. The call to findPart is syntactically correct and earned 1 point. The result is used as a parameter in the call to the ArrayList method add, which was awarded ½ point for being syntactically and semantically correct, even though parts has not been instantiated. The body of the loop is never completed to shorten the parameter in the calls to findPart and the resulting parts ArrayList is not returned.

Sample: A2C
Score: 3

In part (a) the solution attempts to use a traditional for loop {for(int i=0; i<parts.size(); i++)} to traverse parts. Although the loop bounds are correct, individual elements of parts are never accessed so the two ½ points for correctly accessing an element and accessing all elements were not awarded. In addition the ½ point was not awarded for parts.getStart() and parts.getLength() because parts is an ArrayList, not a StringPart. The getStart and getLength methods are instance methods defined in the StringPart class. Since the substring method is not accessed as masterString.substring(...), the 1½ points for extracting a substring from masterString were not awarded. However, since String word is declared, initialized to “”, and the substring extraction attempts concatenated with word, the 1 point for correctly building the string from substrings was awarded. The final ½ point was earned by return word after the loop exit.

This solution earned a total of three of the ½ points in part (b). The ½ point was awarded for declaring and instantiating the parts ArrayList. The instructions for the problem clearly state, “The helper method findPart must be used to choose matching string parts in the master string.” Failure to do this resulted in the loss of 3 whole points for the syntactically correct call of findPart, the calls to findPart involving progressively smaller suffixes of word, and building the correct list of string parts (must have used findPart). However, parts.add(new StringPart(k, length)) is enough to earn the ½ point for the add. Also return parts outside all loops is enough to earn the ½ point for returning an ArrayList of string parts.