2021

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 1

☑ Scoring Guideline

☑ Student Samples

☑ Scoring Commentary

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be* ***unambiguously* *inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does* ***not*** *allow for the reader to assume the use of the lower case variable.*

Question 1: Methods and Control Structures 9 points

Canonical solution

(a) **5 points**

```
public int scoreGuess(String guess)
{
   int count = 0;

   for (int i = 0; i <= secret.length() - guess.length(); i++)
   {
      if (secret.substring(i, i + guess.length()).equals(guess))
      {
         count++;
      }
   }

   return count * guess.length() * guess.length();
}
```

(b) **4 points**

```
public String findBetterGuess(String guess1, String guess2)
{
   if (scoreGuess(guess1) > scoreGuess(guess2))
   {
      return guess1;
   }
   if (scoreGuess(guess2) > scoreGuess(guess1))
   {
      return guess2;
   }
   if (guess1.compareTo(guess2) > 0)
   {
      return guess1;
   }
   return guess2;
}
```

(a) `scoreGuess`

	Scoring Criteria	Decision Rules	
1	Compares `guess` to a substring of `secret`	Responses **can** still earn the point even if they only call `secret.indexOf(guess)` Responses **will not** earn the point if they use `==` instead of `equals`	**1 point**
2	Uses a substring of `secret` with correct length for comparison with `guess`	Responses **can** still earn the point even if they • only call `secret.indexOf(guess)` • use `==` instead of `equals`	**1 point**
3	Loops through all necessary substrings of `secret` (*no bounds errors*)	Responses **will not** earn the point if they skip overlapping occurrences	**1 point**
4	Counts number of identified occurrences of `guess` within `secret` (*in the context of a condition involving both* `secret` *and* `guess`)	Responses **can** still earn the point even if they • initialize count incorrectly or not at all • identify occurrences incorrectly	**1 point**
5	Calculates and returns correct final score (*algorithm*)	Responses **will not** earn the point if they • initialize count incorrectly or not at all • fail to use a loop • fail to compare `guess` to multiple substrings of `secret` • count the same matching substring more than once • use a changed or incorrect `guess` length when computing the score	**1 point**
		Total for part (a)	**5 points**

(b) `findBetterGuess`

	Scoring Criteria	Decision Rules	
6	Calls `scoreGuess` to get scores for `guess1` and `guess2`	Responses **will not** earn the point if they • fail to include parameters in the method calls • call the method on an object or class other than `this`	**1 point**
7	Compares the scores	Responses **will not** earn the point if they • only compare using `==` or `!=` • fail to use the result of the comparison in a conditional statement	**1 point**
8	Determines which of `guess1` and `guess2` is alphabetically greater	Responses **can** still earn the point even if they reverse the comparison Responses **will not** earn the point if they • reimplement `compareTo` incorrectly • use result of `compareTo` as if `boolean`	**1 point**
9	Returns the identified `guess1` or `guess2` (*algorithm*)	Responses **can** still earn the point even if they • call `scoreGuess` incorrectly • compare strings incorrectly Responses **will not** earn the point if they • reverse a comparison • omit either comparison • fail to return a guess in some case	**1 point**
		Total for part (b)	**4 points**

Question-specific penalties

None

Total for question 1 **9 points**

Question 1	Question 2	Question 3	Question 4
●	○	○	○

Begin your response to each question at the top of a new page.

```
public int scoreGuess (String guess)
{

    int count = 0;
    String mod = secret;
    while ( mod.indexOf(guess) >= 0) {
        count++;
        mod = mod.substring(mod.indexOf(guess) +1);
     }
   return count • guess.length();
  }
```

Q1 Sample A 2 of 2

Question 1	Question 2	Question 3	Question 4
●	●	○	○

Begin your response to each question at the top of a new page.

```
Public String findBetterGuess(string guess1,
 string guess2) {
   int g1 = scoreGuess(guess1);
   int g2 = scoreGuess(guess2);

    if(g1 > g2)
      return guess1;
    else if(g2 > g1)
      return guess2;
    else
    {
         if(guess1.compareTo(guess2) > 0)
           return guess1;
         else
          return guess2;
       }
    }
```

Question 1 ● Question 2 ○ Question 3 ○ Question 4 ○

Begin your response to each question at the top of a new page.

a)

```
public int scoreGuess(string guess)
{ int count = 0;
  String temp;
  for(int x = 0; x <= secret.length(); x++)
     { if(secret.indexof(guess) >= 0)
         { count = count + 1;
           temp = secret.substring(secret.indexof(guess));
           }
         }
       int score = 0;
       Score = count + guess.length() * guess.length();
       return score;
       }
```

Question 1	Question 2	Question 3	Question 4
●	○	○	○

Begin your response to each question at the top of a new page.

```
b) public string findBetter Guess( string guess1, String guess2)
 { string bGuess;
     if( scoreGuess(guess1) > scoreGuess(guess2)
      { bGuess = guess1;}
     else if( scoreGuess(guess1) < scoreGuess(guess2)
       { bGuess = guess2; }

     else {
       if  (guess1.compare 2 < 0)
          { return: guess 2 }
        else{
          return guess 1}
          }
      return bGuess;
      }
```

Question 1 ● **Question 2** ○ **Question 3** ○ **Question 4** ○

Begin your response to each question at the top of a new page.

a)

```
public int scoreGuess (String guess){
    int occurrences;
    if( secret.indexOf( guess) < 0){
     scoreGuess = 0;
     }
   else if (secret.indexOf (guess) > 0){
secret = secret. Substring (secret. indexOf(guess)
     occurrences ++;
    }
  return occurrences * guess. length() * guess. length();
  }
```

b)

```
 public String findBetterGuess (String guess 1, String guess2){
  if (game. scoreGuess (guess 1) > game. scoreGuess (guess 2){
       findBetterGuess = guess 1;
    }
  if (game .scoreGuess (guess 2) >game. scoreGuess (guess 1){
      find BetterGuess = guess 2;
   }
else if (game. ScoreGuess (guess 1) = game. scoreGuess(guess2){
        If (guess1. compareTo (guess 2) >0){
           findBetterGuess = guess 1;
         }
        if (guess1. compareTo (guess 2)<0){
           findBetterGuess = guess 2;
        }
   else if (guess1.compareTo(guess2) =0){
          FindBetterGuess = guess 1;
        }
        }
```

Question 1

Overview

This question tested the student's ability to:

- Write program code to create objects of a class and call methods.
- Write program code to satisfy methods using expressions, conditional statements, and iterative statements.

More specifically, this question assessed the ability to use `String` objects, iterate through a range, call `String` methods, and use a method's return value in a conditional expression.

In part (a) students were asked to loop through substrings of `secret` to determine whether there is an occurrence of the string `guess` within `secret`. Students accumulated a count of the number of occurrences of `guess` within `secret`. They were expected to initialize a numeric counter, iterate through all the substrings of `secret`, and update the counter. The students then had to calculate the return value, which is the product of their counter and the square of the length of `guess`.

In part (b) students were asked to compare the results of a method call using conditional statements. They needed to test which return value from two calls to `scoreGuess` was greater and return the parameter with the higher return value. The students also needed to perform an alphabetical comparison of the two parameters if the return values from the `scoreGuess` method calls were equal. They needed to return the correct string based on their comparisons.

Sample: 1A
Score: 8

In part (a) point 1 was earned by calling `indexOf` on `mod`, with `guess` as a parameter. The variable `mod` is initially a reference to `secret` and later contains substrings of `secret`. The point is earned because `indexOf` effectively does a comparison between `secret` and `guess` to determine the position of the first occurrence of `guess` in `secret`. The `String mod` can be modified without destroying the persistent data stored in `secret`. Point 2 was earned by calling `indexOf(guess)` on a reference to `secret`. Point 3 was earned by looping through all necessary substrings of `mod` by creating a substring that begins at the index of the found `guess` plus 1. Point 4 was earned by counting identified occurrences of `guess` within `secret` in the context of a condition and within a `while` loop.

In part (b) point 5 was not earned because the returned value is `count * guess.length()` instead of the product of `count` and the square of `guess.length()`, although the count was correctly computed. Note that using the dot instead of an asterisk is not, by itself, a problem; using common mathematical symbols such as • for multiplication is one of the minor errors for which no penalty is assessed. (See the "No Penalty" category on page 1 of the Scoring Guidelines for a complete list.) Point 6 was earned by the two correct calls to the `scoreGuess` method with `guess1` and `guess2` as parameters. Point 7 was earned by testing if `g1`, the response's local variable storing the score of `guess1`, is greater than `g2`, the response's local variable storing the score of `guess2`. Point 8 was earned because the `compareTo` method is called correctly to compare the two guesses, and its result is compared to zero. Point 9 was earned because the response returns the correctly identified `guess1` or `guess2` in all required cases.

Question 1 (continued)

Sample: 1B
Score: 6

In part (a) point 1 was earned by comparing `guess` to `secret` using `indexOf`. Point 2 was earned by comparing `guess` to `secret` using `indexOf`. Point 3 was not earned because the response does not loop through all necessary substrings of `secret`. The variable `temp` is assigned a substring in the loop, as in one common solution strategy, but the substring and starting index are taken from the original value of `secret` rather than from `temp`, so the same substring is compared repeatedly. Point 4 was earned by counting the number of identified occurrences of `guess` within `secret`. Point 5 was not earned because the algorithm adds `count` to the square of `guess.length()` instead of multiplying.

In part (b) point 6 was earned because the response calls the `scoreGuess` method correctly with `guess1` and `guess2` as parameters. Point 7 was earned because the response compares the return values of the two `scoreGuess` method calls. Point 8 was not earned because the method `compare2` does not exist. Point 9 was earned because each comparison returns the identified `guess1` or `guess2`. The incorrect comparison from point 8 does not affect point 9 because the implied logic of the alphabetical comparison is correct.

Sample: 1C
Score: 4

In part (a) point 1 was earned by comparing `guess` to `secret` using `indexOf`. Point 2 was earned by comparing `guess` to `secret` using `indexOf`. A penalty (-1y) was applied because the response modifies the value of `secret`. Responses should not destroy persistent data (e.g., modifying a `private` instance variable). Point 3 was not earned because the response does not include a loop. Point 4 was earned because the response increments a counter within the context of a conditional involving `secret` and `guess`. Without a loop, the response can identify at most one occurrence of `guess` within `secret`, even if other occurrences exist. Point 5 was not earned because the response does not include a loop.

In part (b) point 6 was not earned because the response calls the `scoreGuess` method on `game`, which is an object or class other than `this`. Point 7 was earned by comparing the results of the `scoreGuess` method calls. Point 8 was earned by determining whether `guess1` or `guess2` is alphabetically greater. Point 9 was not earned because the response does not include a `return` statement.