2019

AP® CollegeBoard

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 3

☑ Scoring Guideline

☑ Student Samples

☑ Scoring Commentary

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Apply the question assessment rubric 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

- o Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- o Spelling/case discrepancies where there is no ambiguity*
- o Local variable not declared provided other variables are declared in some part
- o `private` or `public` qualifier on a local variable
- o Missing `public` qualifier on class or constructor header
- o Keyword used as an identifier
- o Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- o `[]` vs. `()` vs. `<>`
- o `=` instead of `==` and vice versa
- o `length`/`size` confusion for array, `String`, `List`, or `ArrayList`; with or without `()`
- o Extraneous `[]` when referencing entire array
- o `[i,j]` instead of `[i][j]`
- o Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- o Missing `;` where structure clearly conveys intent
- o Missing `{ }` where indentation clearly conveys intent
- o Missing `( )` on parameter-less method or constructor invocations
- o 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 3: Delimiters

Part (a)	`getDelimitersList`	**4 points**

Intent: *Store delimiters from an array in an* `ArrayList`

+1 Creates `ArrayList<String>`

+1 Accesses all elements in array `tokens` (*no bounds errors*)

+1 Compares strings in `tokens` with both instance variables (*must be in the context of a loop*)

+1 Adds delimiters into `ArrayList` in original order

Part (b)	`isBalanced`	**5 points**

Intent: *Determine whether open and close delimiters in an* `ArrayList` *are balanced*

+1 Initializes accumulator(s)

+1 Accesses all elements in `ArrayList delimiters` (*no bounds errors*)

+1 Compares strings in `delimiters` with instance variables and updates accumulator(s) accordingly

+1 Identifies and returns appropriate `boolean` value to implement one rule

+1 Identifies and returns appropriate `boolean` values for all cases

Question 3: Scoring Notes

Part (a) `getDelimitersList`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they…	Responses will not earn the point if they…
+1	Creates `ArrayList<String>`	• omit `<String>`	• omit the keyword `new`
+1	Accesses all elements in array `tokens` (*no bounds errors*)	• return incorrectly inside the loop	• treat `tokens` as a single string • access elements of `tokens` as if from an `ArrayList` (e.g., `tokens.get(i)`)
+1	Compares strings in `tokens` with both instance variables (*must be in the context of a loop*)	• access elements of `tokens` as if from an `ArrayList` (e.g., `tokens.get(i)`)	• use `==` for string comparison • treat `tokens` as a single string
+1	Adds delimiters into `ArrayList` in original order	• add a delimiter by accessing `tokens` incorrectly (e.g., `tokens.get(i)`)	• add a token that is not a delimiter • do not maintain the original delimiter order
Part (b) `isBalanced`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they…	Responses will not earn the point if they…
+1	Initializes accumulator(s)	• initialize inside the loop	• initialize an accumulator variable but don't update it
+1	Accesses all elements in `ArrayList delimiters` (*no bounds errors*)	• return incorrectly inside the loop	• access elements of `delimiters` as if from an array (e.g., `delimiters[i]`)
+1	Compares strings in `delimiters` with instance variables and updates accumulator(s) accordingly	• access elements of `delimiters` as if from an array (e.g., `delimiters[i]`)	• use `==` for string comparison • adjust an accumulator without a guarding condition
+1	Identifies and returns appropriate `boolean` value to implement one rule	• check for more closing delimiters (inside a loop) and return `false` • return `true` if the number of open and close delimiters is the same, and `false` otherwise (after a loop)	
+1	Identifies and returns appropriate `boolean` values for all cases	• have correct logic with the exception of a loop bounds error, accessing elements as if from an array, or using `==` for string comparison	• initialize accumulator inside a loop • fail to check for more closing delimiters inside a loop

Question 3: Delimiters

Part (a)

```
public ArrayList<String> getDelimitersList(String[] tokens)
{
   ArrayList<String> d = new ArrayList<String>();
   for (String str : tokens)
   {
      if (str.equals(openDel) || str.equals(closeDel))
      {
         d.add(str);
      }
   }
   return d;
}
```

Part (b)

```
public boolean isBalanced(ArrayList<String> delimiters)
{
   int openCount = 0;
   int closeCount = 0;

   for (String str : delimiters)
   {
      if (str.equals(openDel))
      {
         openCount++;
      }
      else
      {
         closeCount++;
      }

      if (closeCount > openCount)
      {
         return false;
      }
   }

   if (openCount == closeCount)
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

Complete method `getDelimitersList` below. 3Aa

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
{
    ArrayList<String> result = new ArrayList<String>();
    for (String str : tokens){
      if (str.equals(openDel) || str.equals(closeDel))
          result.add(str);
    }
    return result;
 }
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE.

Complete method isBalanced below. 3Ab

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
{
    int open=0;
    for (int i=0; i<delimiters.size(); i++){
        if (delimiters.get(i).equals(openDel))
            open++;
        else
            open--;
        if (open<0)
            return false;
    }
    return open==0;
}
```

GO ON TO THE NEXT PAGE.

3Ba

Complete method `getDelimitersList` below.

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
[
  ArrayList <String> delims =new ArrayList<String>;
  for(int i=0; i< tokens.length-1; i++)
  [
    if ((tokens[i].equals(openDel))||(tokens[i].equals(closeDel)))
    [
      delims.add(tokens[i]);
    ]
  ]
  return delims;
]
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE.

Complete method isBalanced below. 3Bb

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
[
  int oDel = 0;
  int cDel = 0;
  For(int i=0; i< delimiters.size()-1; i++)
  [
    if(delimiters[i].equals(openDel))
    [
        oDel++;
    ]
    else
    [
       cDel++;
    ]
  ]
  if(oDel != cDel)
  [
    return False;
  ]
  else
  [ return true;

  ]
]
```

GO ON TO THE NEXT PAGE.

Complete method getDelimitersList below. 3Ca

```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

```
ArrayList<String> delimiters;
for(int i = 0; i < tokens.length; i++) {
    if(tokens[i] == openDel) {
        delimiters.add(tokens[i]);
    }
    if(tokens[i] == closeDel) {
        delimiters.add(tokens[i]);
    }
}
return delimeters;
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE.

Complete method `isBalanced` below. 3Cb

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
    int open Count = 0;
    int close Count = 0;

    for (int i = 0; i <= delimiters.size(); i++) {
        if ( delimiters.get(i) = openDel);
            open Count ++;
        }
        if ( delimiters.get(i) = closedDel);
            closeCount ++;
        }
    }
    if ( openCount != closeCount) {
            return false;
    }
    return true;
```

GO ON TO THE NEXT PAGE.

Question 3

Overview

This question tested the student's ability to:

- Write program code to satisfy methods using expressions, conditional statements, and iterative statements; and
- Write program code to create, traverse, and manipulate elements in 1D array or `ArrayList` objects.

This question involved manipulation of both a one-dimensional array and an `ArrayList`, both containing `String` values. Students were expected to write two methods in the enclosing `Delimiters` class, making use of two instance variables of type `String`.

In part (a) students were expected to create an `ArrayList` of `String` objects, then add to it all the values from the given array that matched either of two instance variables. Students had to construct a new `ArrayList` and write a loop to access each element of an array parameter. Inside the loop, students were expected to compare each `String` value in the array with each of the two instance variables and add each matching `String` value to the constructed `ArrayList`.

In part (b) students were given an `ArrayList` containing `String` objects representing open and close delimiters. Students were asked to develop an algorithm to determine whether the given `ArrayList` represents a balanced sequence of open and close delimiters. A sequence is balanced when two conditions are met: (1) When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters. (2) The total number of open delimiters is equal to the total number of close delimiters. Students had to write a loop to access each element of the given `ArrayList`. Inside the loop, students had to compare each `String` value in the `ArrayList` to the instance variables and then update accumulator(s) appropriately.

Sample: 3A
Score: 9

In part (a) the response constructs a new `ArrayList` and assigns it to a local variable. Therefore the response earned point 1. All elements of `tokens` are accessed with an enhanced `for` loop, so point 2 was earned. Each token from the array is compared to both instance variables using the `equals` method, and the response earned point 3. Finally, each identified delimiter is added to the `ArrayList`, and point 4 was earned. Part (a) earned 4 points.

In part (b) the response initializes a variable as an accumulator and updates it later; therefore, it earned point 5. All necessary elements of `delimiters` are accessed with a `for` loop using `ArrayList` notation correctly, so point 6 was earned. Each delimiter from the `ArrayList` is compared to an instance variable using the `equals` method, and the accumulator is updated. Therefore the response earned point 7. The response returns `false` if there are more close delimiters than open delimiters within the loop by determining if the number of open delimiters that have not yet matched a close delimiter ever becomes negative. It also returns `true` after the loop if and only if there are no more open delimiters that have not matched a close delimiter. Because the response returns the correct `boolean` value for both conditions, points 8 and 9 were both earned. Part (b) earned 5 points.

Question 3 (continued)

Sample: 3B
Score: 6

In part (a) the response constructs a new `ArrayList` and assigns it to a local variable. Therefore the response earned point 1. The response was not penalized for missing parentheses when constructing the `ArrayList`. Elements of `tokens` are accessed with a `for` loop using array notation correctly. However, the last element is not accessed, so point 2 was not earned. Each token from the array is compared to both instance variables using the `equals` method; therefore, the response earned point 3. Finally, each identified delimiter is added to the `ArrayList`, and point 4 was earned. Part (a) earned 3 points.

In part (b) the response initializes two variables as accumulators and updates them later; therefore, it earned point 5. Elements of `delimiters` are accessed with a `for` loop. However, the last element is not accessed, and elements are accessed as if from an array, so point 6 was not earned. Each delimiter from the `ArrayList` is compared to an instance variable using the `equals` method, and the accumulators are updated. Therefore the response earned point 7. After the loop terminates, the response tests that the number of open and close delimiters is the same, and the correct `boolean` value is returned in both cases. Because the response correctly implements one of the two conditions specified in the question, it earned point 8. However, it did not earn point 9 because it omits the test for more close delimiters than open delimiters inside the loop. Part (b) earned 3 points.

Sample: 3C
Score: 3

In part (a) the response declares but does not construct a new `ArrayList`, so point 1 was not earned. The loop goes out of bounds by accessing an element after the last valid index of the `tokens` array, so point 2 was not earned. Point 3 was not earned because the response uses the `==` operator to compare the strings in `tokens` with instance variables. Finally, each identified delimiter is added to the `ArrayList`, and so point 4 was earned. Part (a) earned 1 point.

In part (b) the response initializes a variable as an accumulator and updates it later; therefore, it earned point 5. The loop goes out of bounds by accessing an element after the last valid index of `delimiters`, and so point 6 was not earned. Point 7 was not earned because the response uses the `==` operator to compare the strings in `delimiters` with instance variables. After the loop terminates, the response tests that the number of open and close delimiters is the same, and the correct `boolean` value is returned in both cases. Because the response correctly implements one of the two conditions specified in the question, it earned point 8. However, it did not earn point 9 because it omits the test for more close delimiters than open delimiters inside the loop. Part (b) earned 2 points.