2017

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
2017 GENERAL 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

- 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.*

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 3: PhraseEditor

Part (a)	`replaceNthOccurrence`	**5 points**

Intent: *Replace the nth occurrence of a given string with a given replacement*

+1 Calls `findNthOccurrence` to find the index of the `nth` occurrence

+1 Preserves `currentPhrase` only if `nth` occurrence does not exist

+1 Identifies components of `currentPhrase` to retain (uses `substring` to extract before/after)

+1 Creates replacement string using identified components and `repl`

+1 Assigns replacement string to instance variable (`currentPhrase`)

Part (b)	`findLastOccurrence`	**4 points**

Intent: *Return the index of the last occurrence of a given string*

+1 Calls `findNthOccurrence` to find the index of the *n*th occurrence

+1 Increments (or decrements) the value used as *n* when finding *n*th occurrence

+1 Returns the index of the last occurrence, if it exists

+1 Returns -1 only when no occurrences exist

Question-Specific Penalties

-1 (q) Uses `currentPhrase.findNthOccurrence`

-2 (r) Confused identifier instead of `currentPhrase`

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 3: Scoring Notes

Part (a) `replaceNthOccurrence`			**5 points**
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Calls `findNthOccurrence` to find the index of the `nth` occurrence	• do not use the result of calling `findNthOccurrence`	
+1	Preserves `currentPhrase` only if `nth` occurrence does not exist		• fail to use a conditional
+1	Identifies components of `currentPhrase` to retain (uses `substring` to extract before/after)	• identify start and end of substring to be replaced	
+1	Creates replacement string using identified components and `repl`		• create a replacement string that is out of order
+1	Assigns replacement string to instance variable (`currentPhrase`)		
Part (b) `findLastOccurrence`			**4 points**
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Calls `findNthOccurrence` to find the index of the *n*th occurrence	• do not use the result of calling `findNthOccurrence`	• `return currentPhrase.lastIndexOf(str);` • call `findNthOccurrence` with an integer parameter of 0
+1	Increments (or decrements) the value used as *n* when finding *n*th occurrence	• `return currentPhrase.lastIndexOf(str);` • advance through `currentPhrase` searching for *n*th occurrence of `str`	
+1	Returns the index of the last occurrence, if it exists	• `return currentPhrase.lastIndexOf(str);` • compute the correct value to be returned in all cases, but no return statement exists for any case	• shorten string being searched • always return in first iteration of the loop
+1	Returns -1 only when no occurrences exist	• `return currentPhrase.lastIndexOf(str);`	• compute the correct value to be returned in all cases, but no return statement exists for any case • always return in first iteration of the loop

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 3: PhraseEditor

Part (a)

```
public void replaceNthOccurrence(String str, int n, String repl)
{
      int loc = findNthOccurrence(str, n);

      if (loc != -1)
      {
            currentPhrase = currentPhrase.substring(0, loc) + repl +
                            currentPhrase.substring(loc + str.length());
      }
}
```

Part (b)

```
public int findLastOccurrence(String str)
{
      int n = 1;
      while (findNthOccurrence(str, n+1) != -1)
      {
            n++;
      }
      return findNthOccurrence(str, n);
}
```

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.

3Aa

The `Phrase` class includes the method `findNthOccurrence`, which returns the nth occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 *  If the nth occurrence does not exist, the current phrase is unchanged.
 *  Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
```

```
public void replaceNthOccurrence(String str, int n, String repl){
    int index = findNthOccurence(str, n);
    if(index != -1){
        String str2 = currentPhrase.substring(index + str.length());
        String replaced = currentPhrase.substring(0,index) + repl + str2;
        currentPhrase = replaced;
    }
}
```

Part (b) begins on page 16.

3Ab

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;
 *  returns -1 if str is not found.
 *  Precondition: str.length() > 0
 *  Postcondition: the current phrase is not modified.
 */
public int findLastOccurrence(String str)
```

```
public int findLastOccurrence(String str){
    int n=0
    int index = findNthOccurence(str,n+1);
    while(index!=-1){
        n++;
        index = findNthOccurence(str, n+1);
    }
    if(n!=0){
        return findNthOccurence(str, n);
    }else{
        return -1;
    }
}
```

3Ba

The `Phrase` class includes the method `findNthOccurrence`, which returns the nth occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 *  If the nth occurrence does not exist, the current phrase is unchanged.
 *  Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
{
   if (currentPhrase.findNthOccurrence(str, n) == -1)
   {
      return;
   }
   int index = currentPhrase.findNthOccurrence(str, n);
   temp1 = currentPhrase.substring(0, index)
   temp2 = currentPhrase.substring(index, index + (str.length))
   temp3 = currentPhrase.substring(index + (str.length))
   temp2 = repl;
   str = temp1 + temp2 + temp3;

}
```

Part (b) begins on page 16.

3Bb

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;
 *  returns -1 if str is not found.
 *  Precondition: str.length() > 0
 *  Postcondition: the current phrase is not modified.
 */
public int findLastOccurrence(String str)
{
    int count = 0
  for (int i = 1; i <= str.length; i++)
  {
     if (currentPhrase.findNthOccurrence(str, i) == -1 && count == 0)
        return -1;
     count ++;
     if (currentPhrase.findNthOccurrence(str, i) == -1 && count >= 1)
        break;
  }
  return (currentPhrase.findNthOccurrence(str, count));
}
```

3Ca

The `Phrase` class includes the method `findNthOccurrence`, which returns the nth occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 *  If the nth occurrence does not exist, the current phrase is unchanged.
 *  Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
{
   int pos = findNthOccurence(String str, int n);
   if (pos != -1)
   {
      p.substring(pos, str.length()) = repl;
   }
}
```

Part (b) begins on page 16.

3Cb

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;
 *  returns -1 if str is not found.
 *  Precondition: str.length() > 0
 *  Postcondition: the current phrase is not modified.
 */
public int findLastOccurrence(String str)
{
    int val = 0;
    for (n=0; n<p.length(); n++)
    {
        if (p.indexOf(str) != -1)
        {
            val= findNthOccurence(str, n);
        }
    }
    return val;
```

AP® COMPUTER SCIENCE A
2017 SCORING COMMENTARY

Question 3

Overview

This question tested the students' ability to use `String` methods from the AP Java subset to perform processing of strings using various parameters and instance variables. The problem required students to use a provided helper method in their solutions.

In part (a) students were asked to write a method to examine and potentially modify the instance variable `currentPhrase`. Students were required to use the already-implemented `findNthOccurrence` helper method to replace the `nth` occurrence of a string in `currentPhrase` if it was present the number of times specified by the parameter. The new string was created by identifying and extracting the substrings of `currentPhrase` to retain, concatenating these strings with the replacement string parameter `repl` in the correct order, and assigning this value to `currentPhrase`.

In part (b) students were asked to write a method to find the index of the last occurrence of a specified string in `currentPhrase` using the already-implemented `findNthOccurrence` helper method. In finding the last occurrence, student solutions need to be capable of examining `currentPhrase` multiple times using either a call to `findNthOccurrence` or by reimplementing this functionality and examining various substrings while advancing through `currentPhrase`.

Students who used `findNthOccurrence` to examine `currentPhrase` were required to demonstrate an understanding of iteration: setting the loop lower bound ensuring the precondition $n > 0$ is not violated in the `findNthOccurrence` call and setting the loop upper bound to allow `findNthOccurrence` to be called with an argument `n` equal to `currentPhrase.length`.

Students who reimplemented the `findNthOccurrence` functionality were also required to demonstrate an understanding of iteration. The loop bounds needed to be set to ensure that no bounds errors occurred in the matching of substrings to the string parameter.

Regardless of the algorithm used to find the *n*th occurrence, students were required to return the correct index of the last occurrence, or return -1 if no occurrence was found.

Sample: 3A
Score: 9

In part (a) the response earned point 1 because `findNthOccurrence` is correctly called with the correct arguments. Point 2 was earned because the instance variable `currentPhrase` is preserved only when the `nth` occurrence does not exist. The correct components of `currentPhrase` to retain are identified, so point 3 was earned. The replacement string is correctly created, which earned point 4, and the replacement string is assigned to `currentPhrase`, which earned point 5. Part (a) earned 5 points.

In part (b) point 1 was earned by calling `findNthOccurrence` with the proper arguments that satisfy the precondition requiring *n* to be greater than zero. Point 2 was earned in the `while` loop when the `int` argument `(n)` used in the call to `findNthOccurrence` is incremented when attempting to find the next occurrence of `str`. While `findNthOccurrence` returns a value not equal to -1, the loop continues to search for the next occurrence. The loop terminates when `findNthOccurrence(str, n+1)` returns -1, with `n` storing the correct number of occurrences of `str`. If there is at least one occurrence of `str`, the solution correctly returns the index of the last occurrence using a call to `findNthOccurrence(str, n)`, so point 3 was earned. If no occurrences of `str` exist, the response correctly returns -1 and earned point 4. Part (b) earned 4 points.

Question 3 (continued)

Sample: 3B
Score: 6

In part (a) the response calls `findNthOccurrence` on `currentPhrase`. The object `currentPhrase` is not a `Phrase` object, so this is an incorrect method call. Because the response uses the same call `currentPhrase.findNthOccurrence(str, n)` in parts (a) and (b), this response earned point 1 but a one-point question-specific penalty was deducted. This response preserves `currentPhrase` if `currentPhrase` does not contain `n` occurrences of `str`, so it earned point 2. The response identifies the correct components of `currentPhrase` to retain, which earned point 3. The response creates the replacement string, using identified components and `repl`, and earned point 4. The response did not earn point 5 because `currentPhrase` is not assigned the value of the replacement string. Part (a) earned 4 points, but a one-point question-specific penalty was deducted.

In part (b) the response calls `currentPhrase.findNthOccurrence(str, n)` with valid arguments and earned point 1 because the question-specific penalty was already deducted in part (a). Within the bounds of the `for` loop, the variable `i` is incremented and used in the call to `findNthOccurrence`, so point 2 was earned. This response did not earn point 3 because it fails to return the index of the last occurrence, if it exists, because the loop bounds only go until `str.length`, not `currentPhrase.length`. Point 4 was earned because the response correctly returns -1 if no occurrences of `str` exist. Part (b) earned 3 points.

Sample: 3C
Score: 2

In part (a) this response's call to `findNthOccurrence` includes parameter type and did not earn point 1. Point 2 was earned because the return value from the call to `findNthOccurrence` is used to preserve `currentPhrase` only if the `nth` occurrence does not exist. Point 3 was not earned as this response fails to identify the components of `currentPhrase` to retain. Point 4 was not earned as no replacement string is created using the identified component and `repl`. Point 5 was not earned as the replacement string is not assigned to `currentPhrase`. Part (a) earned 1 point.

In part (b) point 1 was not earned as `findNthOccurrence` is called with `int` parameter equal to `0` in violation of the precondition. Point 2 was earned as the loop increments `n`, the variable used in the call to `findNthOccurrence`. The loop's upper bound is incorrect, so this response did not earn point 3. This response incorrectly returns `0` if no occurrences exist and did not earn point 4. Part (b) earned 1 point.