Chief Reader Report on Student Responses:

2024 AP® Computer Science A Free-Response Questions

- Number of Students Scored 98,136
- Number of Readers 494
- Score Distribution

Exam Score	N	%At
5	25,137	25.6
4	21,038	21.4
3	19,754	20.1
2	10,613	10.8
1	21,594	22.0

- Global Mean 3.18

The following comments on the 2024 free-response questions for AP® Computer Science A were written by the Chief Reader, Don Blaheta, Associate Professor of Computer Science at Longwood University. They give an overview of each free-response question and of how students performed on the question, including typical student errors. General comments regarding the skills and content that students frequently have the most problems with are included. Some suggestions for improving student preparation in these areas are also provided. Teachers are encouraged to attend a College Board workshop to learn strategies for improving student performance in specific areas.

Question 1

Task: Methods and Control
Topic: Bird Feeder
Max Score: 9
Mean Score: 4.83

What were the responses to this question expected to demonstrate?

This question tested the student's ability to:

- Write program code to create objects of a class and call methods (Skill 3.A).
- Write program code to define a new type by creating a class (Skill 3.B).
- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).

More specifically, this question assessed the ability to generate random numbers (both to produce probabilities and to generate an integer within a given range), to call instance methods from other instance methods of the same class, to determine when a proper stopping condition has been met, and to report a counted amount.

In part (a) students were asked to determine the amount of food consumed from a bird feeder depending upon a 5% chance of a bear coming along and eating all the food in the feeder. When a bear is present, all the food is consumed, leaving `currentFood` at `0`. In the other case, the birds eat an amount randomly generated in the integer range from 10 to 50 grams. Should there not be enough food for all the birds, the response should guard against consuming more food than is available, leaving `currentFood` at `0` rather than letting it become negative.

In part (b) students were asked to simulate a period of days where the feeder is visited daily, counting and returning the number of days that the food lasts within a given maximum number of days, `numDays`, and with a given regular group of birds, `numBirds`. Students were expected to consider the possibility of starting the test period with no food, running out of food before the end of the test period, or ending the test period with food remaining.

How well did the responses address the course content related to this question? How well did the responses integrate the skills required on this question?

Write program code to create objects of a class and call methods (Skill 3.A).

Both parts of this question involved calling methods, and most responses did so successfully. In part (a), responses needed to call `Math.random()`, a static non-void method with no parameters. In part (b), responses needed to call `simulateOneDay(numBirds)`, a void method within the same class, with one parameter. In both cases, over two-thirds of responses did so correctly.

Among responses that called the `random` method incorrectly, the most common error was to incorrectly place parameters in the call—apparently in an attempt to control the output range. There are other Java classes, outside the subset described in the Course and Exam Description (CED), that provide that functionality, and a small number of responses used them successfully, earning credit. The exam never

requires the use of such classes and methods; responses that use them incorrectly will not earn the associated points. A small number of responses in part (b) did not include the parameter for the `simulateOneDay` method or passed in something incorrect (e.g., `numDays`). Additionally, a fair number of responses treated the `simulateOneDay` method as if it returned information, when it is defined to be a void method.

Write program code to define a new type by creating a class (Skill 3.B).

Although this skill is primarily tested in Q2 (Class Design), in part (b) this question did involve understanding that providing the result of the computation is done through a `return` statement, and that a method must return a value of the correct type and must return something in all cases.

While a majority of responses did so successfully, a number of them returned in one case (e.g., inside the loop when food runs out) but not another (e.g., when the loop terminates with food still remaining), or the response printed the result instead of returning the result.

Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).

This question primarily tested this skill, both in individual expressions and statements and in their assembly into larger algorithms. Both parts of this question involved using expressions to perform mathematical computations and conditional logic to control the operations, and the second part required iteration. The first part required devising an algorithm involving two unrelated conditions using nested or sequential `if` statements, and the second part required a counting algorithm, modified to avoid miscounting when the feeder is empty.

In part (a), responses needed to implement a 95% probability of birds visiting the feeder versus a 5% probability of a bear emptying it. This was possible to accomplish by a direct comparison of a random value to a constant such as `0.05`, but many responses attempted to first map the random number to a suitable integer range, a slightly more difficult strategy that sometimes led to problems, such as improperly casting or failing to cast. Less frequently, responses used an inappropriate constant or comparison to generate the 5% probability. Responses also needed to randomly choose an integer amount of grams of bird food in the inclusive range `[10, 50]`, which turned out to be the single most difficult aspect of this question, in either part; only about one-third of the responses did so successfully. Casting to an integer was frequently either performed incorrectly or not at all. Other responses had difficulty generating a value in the correct range. While the correct range expression should have been `(int)(Math.random() * 41 + 10)` or the equivalent, many responses miscalculated the range, multiplying by `40`, `50`, or `51`, adding instead of multiplying, or otherwise mis-arranging the required expression. Many responses that handled either random case or both random cases incorrectly were still able to assemble the algorithm correctly (and earn credit for it), but many did not, most commonly by failing to check for or prevent the case where `currentFood` is negative when the method exits.

In part (b), responses needed to loop over the single-day simulation no more than `numDays` times and count the ones that actually update the class variable `currentFood`, while also handling the cases when there is no food at the outset, when there is an abundance of food, and when the feeder dispenses all its food before the end of `numDays` iterations. Responses were generally stronger in part (b), but many had difficulty navigating the complex conditions for terminating the loop and keeping a correct count of days to return. Broadly, most responses did call the simulation method in a loop (although many responses did not

iterate the correct number of times), and most responses checked whether there was food remaining, but a majority of responses had trouble combining those parts into a counting algorithm that didn't over- or undercount. Responses that maintained an explicit counter, which was incremented exactly when the part (a) method was called, were often cleaner and more likely to be correct; responses that used a more general loop counter needed more careful logic and/or arithmetic to compute the count correctly (and many did not do so). In addition, many responses failed to handle the case where the feeder was empty to begin with (or did so incorrectly), and some responses increased the day count each day regardless of whether food was consumed.

What common student misconceptions or gaps in knowledge were seen in the responses to this question?

Common Misconceptions/Knowledge Gaps *Write program code to create objects of a class and call methods.*	*Responses that Demonstrate Understanding*
Some responses incorrectly called `Math.random()` using a parameter when needing to get a range. `Math.random(41)`	`Math.random() * 41`

Some responses called `simulateOneDay` with no parameter or on another object.
`simulateOneDay()`
`food.simulateOneDay(numBirds)`
	`simulateOneDay(numBirds)`

Common Misconceptions/Knowledge Gaps *Write program code to define a new type by creating a class.*	*Responses that Demonstrate Understanding*

Some responses returned a value only in some cases.
`for (...)`
`{`
`  if (currentFood == 0)`
`  {`
`  return count;`
`  }`
`  ...`
`}`
	`for (...)`
	`{`
	`  if (currentFood == 0)`
	`  {`
	`  return count;`
	`  }`
	`  ...`
	`}`
	`return numDays;`

Common Misconceptions/Knowledge Gaps *Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.*	*Responses that Demonstrate Understanding*
Some responses did not correctly consider the precedence of the cast operator when casting a `double` to an `int`. `(int)Math.random() * 100`	`(int)(Math.random() * 100)`
Some responses incorrectly set the range of a random number. `(int)(Math.random() * 101)`	`(int)(Math.random() * 100)`

Some responses computed 5% and 95% probabilities incorrectly.
 	
`int ch = (int)(Math.random() * 100);`
`if (ch <= 5) ...`
 	
OR
 	
`double ch = Math.random();`
`if (ch < 0.5) ...`
	`int ch = (int)(Math.random() * 100);`
	`if (ch < 5) ...`
	
	OR
	
	`double ch = Math.random();`
	`if (ch < 0.05) ...`

Some responses did not guard against leaving a negative quantity of food in the feeder.
 	
`currentFood -= totalEaten;`
	`if (totalEaten > currentFood)`
	`{`
	`  currentFood = 0;`
	`}`
	`else`
	`{`
	`  currentFood -= totalEaten;`
	`}`

Some responses incorrectly set the loop bounds.
 	
`for (int x = 0; x <= numDays; x++)`
	`for (int x = 0; x < numDays; x++)`
	
	OR
	
	`for (int x = 1; x <= numDays; x++)`

Some responses incorrectly used a void method as if it returned a value. `int food = simulateOneDay(numBirds);`	`simulateOneDay(numBirds);`

Some responses always counted the first day, even if there was no food in the feeder to begin with.
 	
`for (int d = 1; d <= numDays; d++)`
`{`
`  simulateOneDay(numBirds);`
`  if (currentFood == 0)`
`  {`
`  return d;`
`  }`
`}`
	`// before loop:`
	`if (currentFood == 0)`
	`{`
	`  return 0;`
	`}`
	
	OR
	
	`for (int d = 0; d < numDays; d++)`
	`{`
	`  if (currentFood == 0)`
	`  {`
	`  return d;`
	`  }`
	`  simulateOneDay(numBirds);`
	`}`

Some responses used a standard loop variable as a day counter without accounting for an off-by-one error or keeping a separate counter.
 	
`for (int j = 0; j < numDays; j++)`
`{`
`  simulateOneDay(numBirds);`
`  if (currentFood == 0)`
`  {`
`  return j;`
`  }`
`}`
	`if (currentFood == 0)`
	`{`
	`  return 0;`
	`}`
	`for (int j = 0; j < numDays; j++)`
	`{`
	`  simulateOneDay(numBirds);`
	`  if (currentFood == 0)`
	`  {`
	`  return j + 1;`
	`  }`
	`}`
	
	OR
	
	`int count = 0;`
	`for (int j = 0; j < numDays; j++)`
	`{`
	`  if (currentFood > 0)`
	`  {`
	`  simulateOneDay(numBirds);`
	`  count++;`
	`  }`
	`}`
	`return count;`

Some responses counted simulation days only if they ended with a nonzero amount of food.
 	
`for (int j = 0; j < numDays; j++)`
`{`
`  simulateOneDay(numBirds);`
`  if (currentFood > 0)`
`  {`
`  count++;`
`  }`
`}`
	`for (int j = 0; j < numDays; j++)`
	`{`
	`  if (currentFood > 0)`
	`  {`
	`  count++;`
	`  }`
	`  simulateOneDay(numBirds);`
	`}`

Based on your experience at the AP® Reading with student responses, what advice would you offer teachers to help them improve student performance on the exam?

- Practice using `Math.random()` to generate integer ranges that do not begin with `0` or `1`.
- Practice using `Math.random()` to determine specified probabilities, and reinforce that this application of randomness does not generally require conversion to `int`.
- If students are inclined to use parts of Java outside the subset described in the CED (for instance, the `Random` class), remind them that unless they are very sure of what they are doing it is safer to only use the Java subset covered in the course. If you teach some of the non-CED parts of Java (as many teachers do, very successfully), make sure the students also are comfortable with the testable classes and methods.
- Emphasize appropriate use of casting and order of operations.
- Practice using concrete examples involving edge case scenarios (e.g., "what if the food starts at `0`", "what if the very first call leaves the food at `0`") to trace and test code to prevent off-by-one errors.

What resources would you recommend to teachers to better prepare their students for the content and skill(s) required on this question?

- Progress checks from units 2, 3, and 4 would be helpful to scaffold students' understanding for the Methods and Control free-response questions.
- The following AP Daily Videos and corresponding Topic Questions can be found in AP Classroom to support this Methods and Control free-response question:
    - Write program code to create objects of a class and call methods. Topics 2.3, 2.4, 2.5, and 2.9.
    - Write program code to define a new type by creating a class. Topic 2.5.
    - Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements. Topics 1.5, 2.9, 3.2, 3.3, 3.4, 4.1, and 4.2.

Question 2

Task: Class Design
Topic: ScoreBoard
Max Score: 9
Mean Score: 5.70

What were the responses to this question expected to demonstrate?

This question tested the student's ability to:

- Write program code to define a new type by creating a class (Skill 3.B).
- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).

This question assessed the ability to design an entire class, with constructor, instance variables, and methods that access and update them; to maintain and update the state of the object according to a specified algorithm; and to build a specified string based on the state of the object.

Students were asked to design a class, `Scoreboard`, to hold and modify data for a game with two teams. This class must contain instance variables to store the names of each team, the scores of each team, and an indicator of which team is current. It also requires a constructor and two methods, each matching a provided specification. In designing the `Scoreboard` class, students were expected to understand and demonstrate how to construct appropriate headers for the class itself and for its instance methods. Students were also expected to understand and correctly use instance variables, including declaration of the variables, initialization of the variables, and use of the variables inside the constructor and methods.

The prompt specifications and example code of the class required students to implement two methods. The first method specified, `recordPlay`, is passed a single parameter representing points scored by the active team. If the point count is positive, then the active team's score variable is increased. If the point count is zero, then the active team is switched to the other team. The method must determine which team is the active team and increase the score for that team only and must successfully update the indicator of the current team, correctly in either direction and only when zero points are scored. The second method, `getScore`, constructs and returns a specified `String` that contains team scores, string literals (hyphens), and the name of the active team (which may already be stored in a variable or may be chosen in a conditional statement).

How well did the responses address the course content related to this question? How well did the responses integrate the skills required on this question?

Write program code to define a new type by creating a class (Skill 3.B).

The headers for the class, the constructor, and both required methods were tightly constrained by the prompt and the examples, and a large majority of responses wrote them correctly, including the `class` keyword, named identifiers, return types, parameters, and declaration as `public`. No error was especially common here, but some responses omitted one or more of the required lines entirely, and some combined the class header with the constructor header, adding parentheses (and sometimes parameters) in the class header itself.

Declaring and initializing instance variables was more challenging, although a majority of responses were still successful at doing so. Most responses successfully declared the instance variables inside the class and

outside of any method or constructor. A few responses omitted the `private` access modifier, while several incorrectly attempted to declare instance variables in the constructor. Most responses correctly initialized the instance variables inside the constructor. Some students declared and initialized instance variables on the same line (inside the class but outside the constructor), which is correct when the initial value is a literal (like `0` or `false`) but will not work to initialize with constructor parameters (such as the team names). Some responses correctly used the `this` keyword to permit the constructor parameter name to match the instance variable name, but others used the same name for constructor parameters and instance variables without adjustment, preventing the initialization from working.

A large majority of the responses accessed and modified the declared instance variables appropriately (e.g., to update the score). A few incorrectly attempted to use the name of the class to access instance variables (e.g., `Scoreboard.team1`). Some responses declared local variables and incorrectly attempted to use them as persistent instance data available to subsequent method calls.

Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).

As is often the case with the class design question, the algorithm implementation is not entirely contained within the method bodies but is also affected by design choices involving the types and initial values of instance variables. In particular, any correct response to this question needs to store the team names and updatable integer scores but can track the active team in a variety of ways, affecting how the active team variable(s) are both accessed and updated. Both methods thus require devising algorithms, one to update scores or switch the active team, and the other to decide which team name to build into the score string.

Successful responses used a variety of approaches to keep track of the active team. Most responses used a `String` or an `int` to track the active team, and a small number of responses used a `boolean` variable. Many responses saved the name of the active team as a `String` and correctly called the `equals` method to compare the active team name to one of the team name instance variables to determine which of the two teams was active—and this approach made the second algorithm simpler, as no conditional statement was required. However, some responses unnecessarily used two `String` variables to simultaneously track the active and inactive teams, and responses that used this approach often failed to update one of the two variables. Some responses used an integer instance variable to track the active team and toggled that integer between two values (e.g., `1` and `2`), similar to the canonical solution shown in the Scoring Guidelines. Some used a `boolean` instance variable, which simplified the update-or-switch algorithm (particularly the “toggle” part) but required that the instance variable be carefully initialized correctly.

Broadly, almost all non-blank responses followed some clear strategy in this respect, and a large majority of responses correctly used a conditional statement to detect that the `recordPlay` parameter’s value was nonzero, update a score if so, and initiate a change of active team otherwise. But nearly half of the responses didn’t assemble the algorithm correctly, either updating the incorrect score (in at least some cases) or failing to switch teams (in at least some cases).

The `getScore` method needed to construct and return an appropriate string. About half of the responses successfully returned the appropriate string. Some incorrect responses chose the team name incorrectly, some constructed a string that did not follow the specification, and some failed to return the constructed string or incorrectly printed the string.

What common student misconceptions or gaps in knowledge were seen in the responses to this question?

Common Misconceptions/Knowledge Gaps *Write program code to define a new type by creating a class.*	*Responses that Demonstrate Understanding*

Some responses used incorrect syntax to declare the class, and some combined the class header and constructor.
 	
`public class Scoreboard()`
 	
OR
 	
`public class Scoreboard(String t1,`
`String t2)`
	`public class Scoreboard`

Some responses failed to declare their instance variables with `private` access or included a different modifier.
 	
`public String t1Name;`
 	
OR
 	
`private static int t1Score;`
 	
OR
 	
`boolean t1isActive;`
	`private String t1Name;`
	
	`private int t1Score;`
	
	`private boolean t1isActive;`

Some responses only declared instance variables for storing the constructor parameters, failing to allocate variables for the scores and active team.
 	
`private String t1Name;`
`private String t2Name;`
	`private String t1Name;`
	`private String t2Name;`
	`private int t1Score;`
	`private int t2Score;`
	`private String activeTeam;`

Some responses did not set the instance data correctly from the constructor's parameters, reversing the assignment statement and thereby assigning the instance variable values to the parameters.
 	
`public Scoreboard(String t1, String t2)`
`{`
`  t1 = t1Name;`
`  t2 = t2Name;`
`}`
	`public Scoreboard(String t1, String t2)`
	`{`
	`  t1Name = t1;`
	`  t2Name = t2;`
	`}`

Some responses incorrectly determined whether the `recordPlay` method's parameter value was zero or positive.
 	
`if (points < 0)`
 	
OR
 	
`if (points >= 0)`
	`if (points == 0)`
	`  // switch active team`
	
	OR
	
	`if (points > 0)`
	`  // add points to active team`
	`else`
	`  // switch active team`

Some responses incorrectly used local variables instead of instance variables to hold a team's score.
 	
`public void recordPlay(int points)`
`{`
`  int scoreT1 = 0;`
`  if (points > 0 && t1Activ)`
`  {`
`  scoreT1 += points;`
`  }`
`  ...`
`}`
	`private int scoreT1;`
	
	`public void recordPlay(int points)`
	`{`
	`  if (points > 0 && t1Activ)`
	`  {`
	`  scoreT1 += points;`
	`  }`
	`  ...`
	`}`

Common Misconceptions/Knowledge Gaps *Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.*	*Responses that Demonstrate Understanding*

Some responses incorrectly changed the active team.
 	
1) active team is set and reset in same call
 	
`if (t1Active == 0)`
`{`
`  t1Active = 1;`
`}`
`if (t1Active == 1)`
`{`
`  t1Active = 0;`
`}`
 	
2) active team is switched for only one team
 	
`if (points > 0)`
`{`
`  t1score += points;`
`}`
`else`
`{`
`  t1Active = 0;`
`}`
 	
3) active team is switched even when the `recordPlay` parameter is positive
 	
`t1Active = !t1Active;`
 	
4) active team is always changed to the same value (instead of toggling)
 	
`t1Active = true;`
	`if (t1Active == 0)`
	`{`
	`  t1Active = 1;`
	`}`
	`else`
	`{`
	`  t1Active = 0;`
	`}`
	
	`if (points > 0)`
	`{`
	`  if (t1Active == 0)`
	`  {`
	`  t1score += points;`
	`  }`
	`  else`
	`  {`
	`  t2score += points;`
	`  }`
	`}`
	`else`
	`{`
	`  if (t1Active == 0)`
	`  {`
	`  t1Active = 1;`
	`  }`
	`  else`
	`  {`
	`  t1Active = 0;`
	`  }`
	`}`
	
	`if (points == 0)`
	`{`
	`  t1Active = !t1Active`
	`}`
	
	`t1Active = !t1Active;`

Some responses failed to build and return the specified string.
 	
1) leaves off the hyphens or builds the string with incorrect data
 	
`return t1Score + t2Score + activeTeamName;`
 	
2) prints to standard output in addition to returning the string or instead of returning the string
 	
`System.out.println(t1Score + "-" + t2Score + "-" + activeTeamName);`
 	
3) incorrectly determines the active team
 	
`if (activeTeam == "Team 1") { return t1Score + "-" + t2Score + "-" + activeTeamName; }`
	`return t1Score + "-" + t2Score + "-" + activeTeamName;`
	
	`return t1Score + "-" + t2Score + "-" + activeTeamName;`
	
	`if (activeTeam.equals(team1Name)) { return t1Score + "-" + t2Score + "-" + activeTeamName; }`

Based on your experience at the AP® Reading with student responses, what advice would you offer teachers to help them improve student performance on the exam?

- Have students write complete FRQs including a class header, constructor, instance data, and methods in addition to modifier and accessor methods. Use previously released FRQs from the College Board AP website.
- Practice with FRQs requiring data design, where the instance variables required to implement correct behavior may be more or different than just the constructor parameters.
- Practice using `boolean` variables to represent two states, toggling from one state to another (e.g., `boolVariable = !boolVariable`).
- Practice concatenating strings with numbers, multiple variables, and literals.
- Practice using compound conditionals with `&&` and `||` operators.

What resources would you recommend to teachers to better prepare their students for the content and skill(s) required on this question?

- Progress checks from units 3 and 5 would be helpful to scaffold students' understanding for the Class Design free-response questions.
- The following AP Daily Videos and corresponding Topic Questions can be found in AP Classroom to support this Class Design free-response question:
  - Write program code to define a new type by creating a class. Topics in unit 5.
  - Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements. Topics 1.3, 3.1, 3.2, and 3.3.

Question 3

Task: Array/ArrayList
Topic: WordChecker
Max Score: 9
Mean Score: 4.33

What were the responses to this question expected to demonstrate?

This question tested the student's ability to:

- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).
- Write program code to create, traverse, and manipulate elements in 1D array or `ArrayList` objects (Skill 3.D).

Students were asked to write two methods of a `WordChecker` class. The class contains an `ArrayList` of `String` objects that is already populated. Students were expected to traverse and manipulate the existing `ArrayList` and construct a new one, and to access and manipulate `String` values.

In part (a) students were asked to write a method that determines whether the stored list represents a "word chain." Doing so requires a traversal capable of accessing all adjacent pairs in the list and comparing those adjacent strings, generally using `indexOf` to determine whether an element of the list contained a previous element. Looking for pairs that do *not* have the property, the method must return `false` when it finds an invalid pair (and `true` at the end if none of the pairs are invalid).

In part (b) students were asked to write an unrelated method. This method needs to construct, build, and return a new `ArrayList` with contents derived from the stored list. To do so, the method must traverse the `wordList` and check if each string element starts with a target string (using either `indexOf` or a guarded call to `substring`), and if so, extract the remainder of the string and add the result to the constructed `ArrayList`.

How well did the responses address the course content related to this question? How well did the responses integrate the skills required on this question?

Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).

In both parts (a) and (b) this skill was assessed specifically with respect to manipulation of `String` objects, with three specific tasks: determining whether one string contains another, determining whether one string starts with another, and extracting the second part of a string after a known prefix. All three proved challenging.

The first two string-related tasks were most straightforwardly solved using the `indexOf` method and an appropriate comparison, but many responses used the method `substring` instead. In part (a) this approach was much more difficult (generally requiring a nested loop to do correctly) and usually caused the responses to not earn the point, but many students were somewhat familiar with the `indexOf` pattern for this one, and about half of the responses were successful. In part (b) the use of `indexOf` was much less common. Using

the `substring` approach required an extra guard on the length of `target` that was frequently forgotten; only about one-fourth of all responses were successful at determining whether the target value was a prefix of the current string. The most successful responses used `indexOf` when identifying whether an element of the list contained `target` at the beginning of the element.

The third string-related task was to extract the "remainder" of a string, typically requiring the use of the `length` method and the use of the one-parameter `substring` method. Many responses were able to remove the correct number of initial characters from a `String`, although unguarded calls to the one-parameter `substring` method were also common and, because they trigger an exception, did not earn the associated point.

Write program code to create, traverse, and manipulate elements in 1D array or `ArrayList` objects (Skill 3.D).

This question primarily tests this skill; some individual points assess narrowly the required skills, and one point in each part assesses the ability to assemble the pieces into a correct algorithm. The first part requires devising an algorithm to identify adjacent pairs with a certain property (though it is related to both the "consecutive pairs" algorithm and the "all elements" algorithm). The second part requires a modified filter algorithm.

Most responses in part (a) used a traditional `for` loop to traverse the elements of the list, but many responses did not construct the bounds correctly to access *consecutive* pairs. By contrast, in part (b), a large majority of responses were successful in the various mechanical aspects: traversing (using either kind of `for` loop), constructing, accessing, and adding to the list.

As expected, assembling the algorithms in each part was a bit more challenging, but for each algorithm roughly half of the responses were successful in identifying what parts were needed and how to piece them together.

What common student misconceptions or gaps in knowledge were seen in the responses to this question?

Common Misconceptions/Knowledge Gaps *Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.*	*Responses that Demonstrate Understanding*

Many responses called `substring` with parameters that would access characters outside the string bounds.
 	
`int len = target.length();`
 	
`String prefix = cur.substring(0, len);`
`String remainder = cur.substring(len);`
`if (prefix.equals(target)) {...}`
	`int len = target.length();`
	`if (cur.length() >= len)`
	`{`
	`  String prefix = cur.substring(0, len);`
	`  String remainder = cur.substring(len);`
	`  if (prefix.equals(target)) {...}`
	`}`

Several responses used the result of `indexOf` incorrectly.
 	
`if (cur.indexOf(target) > 0)`
`  ...`
	`if (cur.indexOf(target) == 0)`
	`  ...  // target` is prefix of `cur`
	
	OR
	
	`if (cur.indexOf(target) != -1)`
	`  ...  // target` is in `cur` somewhere

Several responses attempted to reimplement `indexOf` using `substring` and a loop.
 	
`boolean match = false;`
`for (int j = 0;`
`  j <= cur.length() - prev.length();`
`  j++)`
`{`
`  if (!cur.substring(j,`
`  j + prev.length()).equals(prev))`
`  {`
`  match = true;`
`  }`
`}`
 	
`if (!match)`
`  ...  //` works but very difficult
	`if (current.indexOf(target) == -1)`
	`  ...  //` demonstrates understanding

Some responses called methods that do not exist on `String` objects. `String newStr = cur.remove(target);`	`String newStr =` `  cur.substring(target.length());`

Common Misconceptions/Knowledge Gaps *Write program code to create, traverse, and manipulate elements in 1D array or ArrayList objects.*	*Responses that Demonstrate Understanding*
Many responses accessed elements of an `ArrayList` using array notation. `String current = wordList[i];`	`String current = wordList.get(i);`

Some responses failed to compare adjacent pairs of elements from an `ArrayList`.
 	
`int sz = wordList.size();`
`for (int i = 0; i < sz; i++)`
`{`
`  String cur = wordList.get(i);`
`  String cmp = wordList.get(0);`
`  if (cur.indexOf(cmp) == -1) ...`
`}`
	`int sz = wordList.size();`
	`for (int i = 1; i < sz; i++)`
	`{`
	`  String cur = wordList.get(i);`
	`  String prev = wordList.get(i - 1);`
	`  if (cur.indexOf(prev) == -1) ...`
	`}`

Many responses used bounds that did not account for accessing an adjacent element.
 	
`int sz = wordList.size();`
`for (int i = 0; i < sz; i++)`
`{`
`  String cur = wordList.get(i);`
`  String prev = wordList.get(i - 1);`
`  if (cur.indexOf(prev) == -1) ...`
`}`
 	
OR
 	
`int sz = wordList.size();`
`for (int i = 0; i < sz; i++)`
`{`
`  String next = wordList.get(i + 1);`
`  String cur = wordList.get(i);`
`  if (next.indexOf(cur) == -1) ...`
`}`
	`int sz = wordList.size();`
	`for (int i = 1; i < sz; i++)`
	`{`
	`  String cur = wordList.get(i);`
	`  String prev = wordList.get(i - 1);`
	`  if (cur.indexOf(prev) == -1) ...`
	`}`
	
	OR
	
	`int sz = wordList.size();`
	`for (int i = 0; i < sz - 1; i++)`
	`{`
	`  String next = wordList.get(i + 1);`
	`  String cur = wordList.get(i);`
	`  if (next.indexOf(cur) == -1) ...`
	`}`

Some responses attempted to return inside the loop in both cases (or failed to return a value in some case).

```
for (...)
{
   ...
   if (cur.indexOf(prev) == -1)
   {
      return false;
   }
   else
   {
      return true;
   }
}
```

OR

```
for (...)
{
   ...
   if (cur.indexOf(prev) == -1)
   {
      return false;
   }
}
```

```
for (...)
{
   ...
   if (cur.indexOf(prev) == -1)
   {
      return false;
   }
}
return true;
```

Some responses used an alias of `wordList` (rather than constructing a new list) and then modified it, destroying persistent data.

```
ArrayList<String> result = wordList;
for (int i = 0; i < result.size(); i++)
{
   if (... != 0)
   {
      result.remove(i);
   }
}
```

```
ArrayList<String> result =
   new ArrayList<String>();
for (String word : wordList)
{
   if (... == 0)
   {
      result.add(...);
   }
}
```

Based on your experience at the AP® Reading with student responses, what advice would you offer teachers to help them improve student performance on the exam?

- Reinforce the difference between `ArrayList` objects and arrays, specifically when it comes to accessing elements from each.
- Reinforce creating loop headers to obtain adjacent elements from a collection.
- Encourage students to use available methods from the Quick Reference Guide rather than using methods that don't exist.
- Reinforce how to create an `ArrayList` of objects.
- Continue practicing using `String` methods to find strings in different locations of other strings: at the beginning, at the end, and at various positions in between.

- Continue practicing returning values from methods and emphasize not to print when asked to return a value.
- Continue practicing different algorithms regarding `ArrayList` values.

What resources would you recommend to teachers to better prepare their students for the content and skill(s) required on this question?

- Progress checks from unit 7 would be helpful to scaffold students' understanding for the `ArrayList` free-response questions.
- The following AP Daily Videos and corresponding Topic Questions can be found in AP Classroom to support this Array/`ArrayList` free-response question:
    - Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements. Topics 2.5, 2.7, 3.3, 4.2, and 5.6.
    - Write program code to create, traverse, and manipulate elements in 1D array or `ArrayList` objects. Topics 7.1, 7.2, 7.3, and 7.4.

Question 4

Task: 2D Array
Topic: Grid Path
Max Score: 9
Mean Score: 3.20

What were the responses to this question expected to demonstrate?

This question tested the student's ability to:

- Write program code to create objects of a class and call methods (Skill 3.A).
- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).
- Write program code to create, traverse, and manipulate elements in 2D array objects (Skill 3.E).

This question involved the traversal of a two-dimensional (2D) array of `int` values, accessing variables and methods within the class being written, and creating and accessing instances of a separate, provided class representing grid coordinate pairs. In addition to an unusual traversal pattern, the algorithms required careful understanding of boundaries in a 2D array as well as writing code to guard against out-of-bounds access.

In part (a) students were asked to write a non-void method, `getNextLoc`, which has two integer parameters, `row` and `col`, and returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`. The neighbors to be considered are the grid elements below and to the right of `row` and `col`. The method must verify that a neighbor exists before accessing it to perform a comparison: if only one of the neighbors exists (because the other would be out of bounds), the `Location` of the existing neighbor is returned. The precondition of the method guarantees that `row` and `col` are not the bottom-right corner of the grid, so at least one neighbor always exists.

In part (b) students were asked to write a non-void method, `sumPath`, which computes and returns the sum of all values on a particular path in `grid`. The path begins with the element at the location indicated by parameters `row` and `col` and is determined by successive calls to the `getNextLoc` method written in part (a). The path ends when the element in the last row and the last column of `grid` is reached. Again, the precondition of the method guarantees that `row` and `col` are not the bottom-right corner of the grid, so there will always be at least two positions in the path to traverse; the method must ensure that both the starting value and the ending value are included in the sum (and must not violate any method preconditions along the way).

How well did the responses address the course content related to this question? How well did the responses integrate the skills required on this question?

Write program code to create objects of a class and call methods (Skill 3.A).

In part (a) most responses were able to create `Location` objects. Some responses omitted the `new` keyword in invoking the constructor or failed to create a `Location` object at all. In part (b) most responses were able to call `getNextLoc`, though some responses did so with incorrect syntax (e.g., omitting the

parameters). About half of the responses correctly called the accessor methods `getRow` and `getCol`. Some responses had problems forming syntactically correct method calls, and others made no attempt to call the accessors at all.

Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements (Skill 3.C).

This skill is assessed at a somewhat more sophisticated level here than in other questions. In part (a) guarding the method from accessing or returning any out-of-bounds values requires identifying at least three cases (a complete correct algorithm probably requires four) and building appropriate multi-way `if`/`else` selection to handle it (or possibly a complex compound Boolean expression with similar effect). Less than a third of the responses did so successfully, although most had some kind of conditional structure; the difficulty typically lay in writing expressions to correctly identify the last row or the last column in `grid`. Writing conditionals in the context of a specification involving 2D array boundaries is (as expected) a more challenging application of this skill.

In part (b) responses were generally more successful in declaring, initializing, and updating a sum variable, but still almost half failed to do so. Some omitted the declaration, or did not initialize the local variable, while others incorrectly produced a sum of `Location` objects instead of a sum of integers from `grid`.

Write program code to create, traverse, and manipulate elements in 2D array objects (Skill 3.E).

This is the question that explicitly assesses this skill, both at the basic level of accessing the 2D array and at the more sophisticated level of assembling algorithms to process them. This question required three such algorithms. Part (a) required devising an algorithm to choose the correct neighbor in each of two boundary cases and two non-boundary cases, as well as the building and return of the associated `Location` object. Part (b) required two interacting algorithms: first, an algorithm to find the path, involving repeatedly generating and then unpacking a `Location` object (and stopping at the correct time); and second, a modified sum algorithm where either the first or last element in the sum (or in some designs, both) needs to be specially handled.

When it came to directly accessing an element of the 2D array, the responses were largely successful. A large majority did so correctly in part (a), and about half did so correctly in part (b).

The algorithms required for this question proved very challenging. On the neighbor-choosing algorithm from part (a), about one-third of responses were successful. While miscalculating the boundary condition was assessed elsewhere, a number of responses failed to guard the access at all, treating it as a simple two-way conditional choice (an incorrect choice of algorithm). Others identified the four cases but returned the incorrect value in some of them. On the sum algorithm from part (b), again roughly one-third of responses were successful, and the overwhelming majority of errors came from failing to include either the first or the last element in the computed sum.

The third algorithm, the pathfinding algorithm, was the single most challenging element among all the free-response questions, with about one in eight responses implementing it successfully. Most responses struggled with writing code to correctly traverse `grid`, because the traversal did not follow any traditional pattern. Many responses correctly used a `while` loop controlled by successive calls to `getNextLoc` to traverse `grid` but used an incorrect loop condition. Many responses incorrectly handled moving from

position to position within the grid, even when correctly using `getNextLoc` to determine the next position in the traversal.

A substantial minority of students observed, correctly, that the method in part (b) lent itself very naturally to a recursive implementation. While recursive code is never required in a free-response question, responses that use recursion are certainly permitted, and are eligible for full credit if they work correctly. Among responses using a recursive strategy, most responses were basically successful, although in many of them either the recursive call itself or the call to the other method violated the methods' preconditions.

What common student misconceptions or gaps in knowledge were seen in the responses to this question?

Common Misconceptions/Knowledge Gaps *Write program code to create objects of a class and call methods.*	*Responses that Demonstrate Understanding*

Some responses attempted to use constructors from the `Location` class that were not given or were not called appropriately.
`return new Location();`
OR
`return Location(row, col + 1);`
	`return new Location(row, col + 1);`

Some responses did not create objects of the `Location` class when required. `return grid[row][col + 1];`	`return new Location(row, col + 1);`

Some responses did not call `Location` methods correctly.
`r = Location.getRow();`
OR
`r = grid[row][col].getRow();`
	`r = getNextLoc(row, col).getRow();`

Some responses did not call `getNextLoc` correctly.
 	
`n = GridPath.getNextLoc(r, c);`
 	
OR
 	
`n = grid.getNextLoc(r, c);`
 	
OR
 	
`n = getNextLoc();`
	`n = getNextLoc(r, c);`

Common Misconceptions/Knowledge Gaps *Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.*	*Responses that Demonstrate Understanding*

Some responses failed to correctly determine if `row` was the last row of `grid`.
 	
`if (row == grid.length)`
 	
OR
 	
`if (grid[row + 1][col] == null)`
	`if (row == grid.length - 1)`
	
	OR
	
	`if (row + 1 == grid.length)`

Some responses failed to correctly determine if `col` was the last column of `grid`.
 	
`if (col == grid[0].length)`
 	
OR
 	
`if (grid[row][col + 1] == null)`
	`if (col == grid[0].length - 1)`
	
	OR
	
	`if (col + 1 == grid[0].length)`

Some responses incorrectly produced a sum of `Location` objects instead of integers from `grid`.
 	
`int sum = 0;`
`...`
`sum += getNextLoc(row, col);`
	`int sum = 0;`
	`...`
	`Location loc = getNextLoc(row, col);`
	`sum +=`
	`  grid[loc.getRow()][loc.getCol()];`

Common Misconceptions/Knowledge Gaps *Write program code to create, traverse, and manipulate elements in 2D array objects.*	*Responses that Demonstrate Understanding*

Most responses used an incorrect loop condition to stop the loop when `row` and `col` indicated the bottom-right corner of `grid`.
 	
`while (row < grid.length - 1 && col < grid[0].length - 1)`
 	
OR
 	
`while (row < grid.length && col < grid[0].length)`
 	
OR
 	
`while (grid[row][col] != null)`
	`while (row < grid.length - 1 || col < grid[0].length - 1)`
	
	OR
	
	`while (!(row == grid.length - 1 && col == grid[0].length - 1))`

Many responses incorrectly handled moving from position to position within the grid.
 	
`while (row < grid.length - 1 || col < grid[0].length - 1)`
`{`
`  ...`
`  getNextLoc(row, col);`
`}`
	`while (row < grid.length - 1 || col < grid[0].length - 1)`
	`{`
	`  ...`
	`  Location loc = getNextLoc(row, col);`
	`  row = loc.getRow();`
	`  col = loc.getCol();`
	`}`

Many responses incorrectly handled the requirement to sum *all* the values along the path.
 	
`int sum = 0;`
`while ( ... )`
`{`
`  sum += grid[row][col];`
`  Location loc =`
`  getNextLoc(row, col);`
`  row = loc.getRow();`
`  col = loc.getCol();`
`}`
`return sum;  // omits last value`
 	
OR
	`int sum = 0;`
	`while ( ... )`
	`{`
	`  sum += grid[row][col];`
	`  Location loc =`
	`  getNextLoc(row, col);`
	`  row = loc.getRow();`
	`  col = loc.getCol();`
	`}`
	`return sum + grid[row][col];`
	
	OR

```
int sum = 0;
while ( ... )
{
   Location loc =
      getNextLoc(row, col);
   row = loc.getRow();
   col = loc.getCol();
   sum += grid[row][col];
}
return sum;  // omits first value
```

OR

```
int sum = grid[row][col];
while ( ... )
{
   Location loc =
      getNextLoc(row, col);
   row = loc.getRow();
   col = loc.getCol();
   sum += grid[row][col];
}
return sum + grid[row][col];
   // adds last value twice
```

```
int sum = grid[row][col];
while ( ... )
{
   Location loc =
      getNextLoc(row, col);
   row = loc.getRow();
   col = loc.getCol();
   sum += grid[row][col];
}
return sum;
```

Based on your experience at the AP® Reading with student responses, what advice would you offer teachers to help them improve student performance on the exam?

- Practice accessing an array element near or past the defined dimensions of the array. Reinforce that accessing an array element out-of-bounds does not return `null`, but results in an `ArrayIndexOutOfBoundsException` being thrown. This could be done through either coding assignments or multiple-choice questions.
- Practice accessing an array using different types of traversals other than row-major or column-major order. Consider assigning free-response questions (from a prior year or from AP Classroom) that use non-traditional traversals. Assign open-ended problems where students create their own traversal algorithms through an array.
- When discussing loops, emphasize that the loop condition determines when the loop continues to iterate, not when the loop terminates. Programmers often recognize the loop termination condition first; practice how to convert a termination condition into a continuation condition (e.g., by negating the entire expression or by applying DeMorgan's laws).

What resources would you recommend to teachers to better prepare their students for the content and skill(s) required on this question?

- Progress checks from unit 8 would be helpful to scaffold students' understanding for the 2D array free-response questions.

- The following AP Daily Videos and corresponding Topic Questions can be found in AP Classroom to support this 2D array free-response question:
    - Write program code to create objects of a class and call methods. Topics 2.2 and 2.5.
    - Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements. Topics 3.1, 3.3, 3.4, 3.5, and 4.1.
    - Write program code to create, traverse, and manipulate elements in 2D array objects. Topics 8.1 and 8.2.