2018

AP CollegeBoard

AP Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 1

- ☑ **Scoring Guideline**
- ☑ **Student Samples**
- ☑ **Scoring Commentary**

AP® COMPUTER SCIENCE A
2018 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
2018 SCORING GUIDELINES

Question 1: Frog Simulation

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

Intent: *Simulate the distance traveled by a hopping frog*

+1 Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position

+1 Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)

+1 Determines if a distance representing multiple hops is at least `goalDistance`

+1 Determines if a distance representing multiple hops is less than starting position

+1 Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position

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

Intent: *Determine the proportion of successful frog hopping simulations*

+1 Calls `simulate` the specified number of times (*no bounds errors*)

+1 Initializes and accumulates a count of `true` results

+1 Calculates proportion of successful simulations using `double` arithmetic

+1 Returns calculated value

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 1: Scoring Notes

Part (a) `simulate`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position	• use `hopDistance()` as a position, like `hopDistance() < 0`	• only use `hopDistance()` as a count, like `hopDistance() < maxHops`
+1	Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)		• do not use a loop
+1	Determines if a distance representing multiple hops is at least `goalDistance`	• use some number of hops `*` `hopDistance()` as the frog's final position	
+1	Determines if a distance representing multiple hops is less than starting position		
+1	Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position	• have checks for all three conditions and correct return logic based on those checks, even if a check did not earn a point	• do not check all three conditions • only check for `goalDistance` after the loop • only check for starting position after the loop
Part (b) `runSimulations`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `simulate` the specified number of times (*no bounds errors*)	• do not use the result of calling `simulate`	• do not use a loop
+1	Initializes and accumulates a count of `true` results		• initialize the count inside a loop • do not use a loop
+1	Calculates proportion of successful simulations using `double` arithmetic	• perform the correct calculation on an accumulated value, even if there was an error in the accumulation	• fail to divide by the parameter
+1	Returns calculated value		• calculate values using nonnumeric types • return a count of simulations

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES

Question 1: Frog Simulation

Part (a)

```
public boolean simulate()
{
   int position = 0;

   for (int count = 0; count < maxHops; count++)
   {
      position += hopDistance();
      if (position >= goalDistance)
      {
         return true;
      }
      else if (position < 0)
      {
         return false;
      }
   }
   return false;
}
```

Part (b)

```
public double runSimulations(int num)
{
   int countSuccess = 0;

   for (int count = 0; count < num; count++)
   {
      if(simulate())
      {
         countSuccess++;
      }
   }
   return (double)countSuccess / num;
}
```

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 `simulate` below. You must use `hopDistance` appropriately to receive full credit.

/Aa

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate() {
    int currentDistance = 0;
    for(int i = 0; i < maxHops; i++) {
        currentDistance += hopDistance();
        if(currentDistance < 0) {
            return false;
        }
        if(currentDistance >= goalDistance) {
            return true;
        }
    }
    return false;
}
```

Part (b) begins on page 8

GO ON TO THE NEXT PAGE.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

/AL

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit. •

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num) {
    int numSuccess = 0;
    for(int i = 0; i < num; i++) {
        if(simulate()) {
            numSuccess++;
        }
    }
    return ((double)numSuccess) / num;
}
```

Complete method simulate below. You must use hopDistance appropriately to receive full credit.

1Ba

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
{
  int dist = 0;
  int goal = goalDistance;

  for(int i=0;i<= maxHops;i++)
    {
      dist += hopDistance();
    }
  if(dist >= goal)
    {
      return true;
    }
  else
    {
      return false;
    }
}
```

Part (b) begins on page 8

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`. /6

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
  int count = num;
  int pass = 0;
  double ratio = 0.0;

  for(int i=0;i≤num;i++)
    {
      if(simulate())
        {
          pass++;
        }
    }

  ratio = (double) pass/count;

  return ratio;
}
```

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit. 1Ca

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
```

```
{
  for(k=0; k<=maxHops; k++){
  {
  for(k=0; k<=numHops; k++){
      dist -= hopDistance;
      if(dist>=0){
        return true;
      }
  }
  return false;
}
```

Part (b) begins on page 8

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`. 1/6

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
int count=0;
for(k=0; k<num; k++){
   if(simulate(k)==true){
      count++;
      }

   }
return double (count/num);
}
```

AP® COMPUTER SCIENCE A
2018 SCORING COMMENTARY

Question 1

Overview

This question tested the student's ability to:

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

Students were provided with the specifications of the `FrogSimulation` class. The `FrogSimulation` class encapsulates a simulation of a frog hopping in a straight line. It contains two private integer instance variables, `goalDistance` and `maxHops`, which represent the distance in inches from the starting point to the goal and the maximum number of hops allowed to reach the goal. It also contains a private method, `hopDistance`, which returns an integer representing the distance in inches to be moved when the frog hops. Implementation for this method was not shown.

In part (a) students were asked to write the `FrogSimulation` method `simulate`, which determines whether a frog is successful in reaching `goalDistance`. Students were required to use the private method `hopDistance` within the context of a loop to update an initialized variable representing the frog's position. The loop iterates until one of the following conditions becomes true:

- The frog has reached or passed the goal, in which case a value of `true` is immediately returned.
- The frog has reached a negative position, in which case a value of `false` is immediately returned.
- A frog has taken `maxHops` hops without reaching the goal, in which case a value of `false` is returned.

In part (b) students were asked to write the `FrogSimulation` method `runSimulations(int num)`, which uses a loop to call the `simulate` method `num` times. Each time `simulate` returns `true`, a previously initialized variable is incremented. The method returns a decimal value representing the proportion of simulations in which the frog successfully reached or passed the goal.

Sample: 1A
Score: 9

In part (a) the response earned point 1 by adjusting a variable representing the frog's position with the result of a correct call to `hopDistance`. The variable is properly initialized and is updated within a loop that iterates no more than `maxHops` times. Therefore, the response earned point 2. The response uses conditional statements to correctly determine if the frog's position is greater than or equal to `goalDistance` or if the frog's position is less than the starting position. For this reason, the response earned both points 3 and 4. All of the conditional statements exist within the context of the loop. This allows the response to correctly return `true` if the goal is met and `false` otherwise. As a result the response earned point 5. Part (a) earned 5 points.

In part (b) point 6 is earned because the response implements a loop to correctly call `simulate` exactly `num` times. A properly initialized variable is used to accumulate a count of `true` results, so the response earned point 7. The variable is correctly cast to a `double` value prior to being divided by the parameter. For this reason the response earned point 8. The calculated value is returned, and the response earned point 9. Part (b) earned 4 points.

Question 1 (continued)

Sample: 1B
Score: 5

In part (a) `hopDistance` is called correctly, and the returned result is used to adjust a variable representing the frog's position. An improperly structured assignment operator is used. However, this is a nonpenalized error, and the response earned point 1. Because an incorrect loop terminating condition allows for the frog's position to be updated more than `maxHops` times, the response did not earn point 2. Point 3 was earned when a conditional was used to determine if the frog's position has reached or exceeded `goalDistance`. Because the response does not include a check to determine if the frog's position is less than the starting position, the response did not earn point 4. For multiple reasons, the logic of the return statement is incorrect. First, the `goalDistance` check occurs outside of the loop, so the response could return an incorrect result after the goal has been reached. Second, there is no determination as to whether the distance is less than the starting position. As a result the response could return an incorrect result after a negative position has been achieved. For either of these reasons, the response did not earn point 5. Part (a) earned 2 points.

In part (b) `simulate` is called more than `num` times due to an incorrect terminating condition within the loop. Therefore, the response did not earn point 6. Point 7 was earned because a variable is initialized and correctly used to count `true` results. The variable is correctly cast as a `double` and is divided by a variable containing the parameter value, so the response earned point 8. The calculated value is returned, and the response earned point 9. Part (b) earned 3 points.

Sample: 1C
Score: 3

In part (a) `hopDistance` is called, and the returned result is used to adjust a variable representing distance. Because `hopDistance` is a parameter-less method, the missing parenthesis is a nonpenalized error. Therefore, the response earned point 1. For several reasons, the response does not include the required elements to earn point 2. First, the terminating condition of the loop does not refer to `maxHops` but instead refers to a variable that is neither declared nor initialized. Second, the comparison operator within the terminating condition of the loop allows `hopDistance` to be called more than `maxHops` times. Finally, the variable used to adjust distance is neither declared nor initialized. For any of these reasons, the response did not earn point 2. Because the response does not refer to `goalDistance` in the attempt to determine if the frog reached the goal, the response did not earn point 3. The response does not include a check to determine if the frog's position is less than the starting position, so the response did not earn point 4. Because the starting position check is omitted, the response will not always return a correct value. Therefore, the response did not earn point 5. Part (a) earned 1 point.

In part (b) the response did not earn point 6 because `simulate` is called incorrectly by including a parameter in the method call. A properly initialized variable is used to accumulate a count of `true` values, so the response earned point 7. The calculation of the proportion of successful simulations is incorrectly cast as a `double`, so the response did not earn point 8. The calculated result is returned. Therefore, the response earned point 9. Part (b) earned 2 points.