2017

AP CollegeBoard

AP Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

- ☑ Free Response Question 4
- ☑ Scoring Guideline
- ☑ Student Samples
- ☑ Scoring Commentary

AP Central is the official online home for the AP Program: apcentral.collegeboard.org

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 4: Successor Array

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

Intent: *Find the position of a given integer in a 2D integer array*

+1 Accesses all necessary elements of `intArr` (*no bounds errors*)

+1 Identifies `intArr` element equal to `num` (*in context of an* `intArr` *traversal*)

+1 Constructs `Position` object with same row and column as identified `intArr` element

+1 Selects constructed object when `intArr` element identified; `null` when not

+1 Returns selected value

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

Intent: *Create a successor array based on a 2D integer array*

+1 Creates 2D array of `Position` objects with same dimensions as `intArr`

+1 Assigns a value to a location in 2D successor array using a valid call to `findPosition`

+1 Determines the successor `Position` of an `intArr` element accessed by row and column (*in context of* `intArr` *traversal*)

+1 Assigns all necessary locations in successor array with corresponding position object or `null` (*no bounds errors*)

Question-Specific Penalties

-1 (s) Uses confused identifier `Arr`

-1 (t) Uses `intArr[].length` as the number of columns

-1 (u) Uses non-existent accessor methods from `Position`

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 4: Scoring Notes

Part (a) `findPosition`			**5 points**
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Accesses all necessary elements of `intArr` (*no bounds errors*)		• use `if (...) return; else return null;` inside loop • confuse row and column bounds • fail to traverse `intArr`
+1	Identifies `intArr` element equal to `num` (*in context of an `intArr` traversal*)		• use `.equals` instead of `==`
+1	Constructs `Position` object with same row and column as identified `intArr` element		• omit keyword `new` • use `(r,c)` instead of `Position(r,c)`
+1	Selects constructed object when `intArr` element identified; `null` when not	• use `"null"` instead of `null` • construct a `String` object using row and column indices	• use `if (...) return; else return null;` inside loop • use `(r,c)` instead of `Position(r,c)`
+1	Returns selected value		
Part (b) `getSuccessorArray`			**4 points**
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Creates 2D array of `Position` objects with same dimensions as `intArr`		• omit keyword `new`
+1	Assigns a value to a location in 2D successor array using a valid call to `findPosition`	• call `Successors.findPosition(…)`	• reimplement the code from `findPosition` • call `findPosition` with a single argument • call `this.findPosition(…)`
+1	Determines the successor `Position` of an `intArr` element accessed by row and column *(in context of `intArr` traversal*)	• reimplement the code from `findPosition`	• call `findPosition` using an integer that is not identified with a location in `intArr` • call `findPosition` with a single argument
+1	Assigns all necessary locations in successor array with corresponding position object or `null` (*no bounds errors*)	• use `SuccessorArray` dimensions correctly, even if `SuccessorArray` was not initialized properly • only assign non-null entries to `SuccessorArray`	• reimplement the code from `findPosition` but mishandle the null case. • fail to traverse `intArr`

Return is not assessed in Part (b).

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 4: Successor Array

Part (a)

```
public static Position findPosition(int num, int[][] intArr)
{
      for (int row=0; row < intArr.length; row++)
      {
            for (int col=0; col < intArr[0].length; col++)
            {
                  if (intArr[row][col] == num)
                  {
                        return new Position(row, col);
                  }
            }
      }
      return null;
}
```

Part (b)

```
public static Position[][] getSuccessorArray(int[][] intArr)
{
      Position[][] newArr = new Position[intArr.length][intArr[0].length];

      for (int row=0; row < intArr.length; row++)
      {
            for (int col=0; col < intArr[0].length; col++)
            {
                  newArr[row][col] = findPosition(intArr[row][col]+1, intArr);
            }
      }
      return newArr;
}
```

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.

4Aa

Complete method `findPosition` below.

```
/** Returns the position of num in intArr;
 *  returns null if no such element exists in intArr.
 *  Precondition: intArr contains at least one row.
 */
public static Position findPosition(int num, int[][] intArr)
{ boolean tF = false;
 for(int r=0; r< intArr.length; r++)
 {
   for(int c=0; c< intArr[r].length; c++)
   { if (intArr[r][c] == num) {
       Position p = new Position(r,c);
       fF=true
     }
   }
 }
 if( !tF)
   return null;
 else
   return p;
}
```

Part (b) begins on page 20.

4Ab

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

Complete method `getSuccessorArray` below.

```
/** Returns a 2D successor array as described in part (b) constructed from intArr.
 *  Precondition: intArr contains at least one row and contains consecutive values.
 *                Each of these integers may be in any position in the 2D array.
 */
public static Position[][] getSuccessorArray(int[][] intArr)
{
  Position [][] pos = new Position [intArr.length]
                                   [intArr[0].length];

  for(int r=0; r<intArr.length; r++)
  {
    for(int c=0; c<intArr[r].length; c++)
    {
      pos[r][c] =
            findPosition(intArr[r][c]+1, intArr);
    }
  }
  return pos;
}
```

4Ba

Complete method `findPosition` below.

```
/** Returns the position of num in intArr;
 *  returns null if no such element exists in intArr.
 *  Precondition: intArr contains at least one row.
 */
public static Position findPosition(int num, int[][] intArr)
```

```
for(int i=0; i< intArr.length(); i++){
    for(int j=0; j < intArr[].length(); i++){
        if(intArr[i][j] == num)
            return (2,1);
    }
}
return null;
```

Part (b) begins on page 20.

4B b

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

Complete method `getSuccessorArray` below.

```
/** Returns a 2D successor array as described in part (b) constructed from intArr.
 *  Precondition: intArr contains at least one row and contains consecutive values.
 *                Each of these integers may be in any position in the 2D array.
 */
public static Position[][] getSuccessorArray(int[][] intArr)
    int Position[][] = new Array [intArr.length()] [intArr[].length()];
    for (int i=0, i<intArr.length(), i++){
        for(int j=0, j<intArr[].length(), j++){
            Position[i][j] = findPosition (intArr[i][j]+1,
                                           intArr[][]);
        }
    }
}
```

4Ca

Complete method `findPosition` below.

```
/** Returns the position of num in intArr;
 *  returns null if no such element exists in intArr.
 *  Precondition: intArr contains at least one row.
 */
public static Position findPosition(int num, int[][] intArr)
 {
  for( i=0 ; i< row.length() ; i++)
    for( j=0 ; j< col.length ; j++)
      {
     if( num.equals(int[i][j]))
      {
        return (i,j)
      }
      else
      {
        return null;
      }
    }
}
```

Part (b) begins on page 20.

GO ON TO THE NEXT PAGE.

4C b

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

Complete method `getSuccessorArray` below.

```
/** Returns a 2D successor array as described in part (b) constructed from intArr.
 *  Precondition: intArr contains at least one row and contains consecutive values.
 *                Each of these integers may be in any position in the 2D array.
 */
public static Position[][] getSuccessorArray(int[][] intArr)
    {
      Successor = intArr + 1 ;
      if( 5 < successor && successor < 17)
        {
          return successor.findPosition();
        }
      else
        {
          return null;
        }
    }
```

AP® COMPUTER SCIENCE A
2017 SCORING COMMENTARY

Question 4

Overview

This question involved reasoning about two-dimensional (2D) arrays of integers and objects. The students were expected to write two static methods of an enclosing `Successors` class. A provided `Position` class was used to represent an integer's location (row and column) in a 2D array.

In part (a) students were asked to implement a static method with two parameters, an integer value, and a 2D array of integers. They were expected to search the given array for the given value. If found, students were expected to create and return a new `Position` object representing the value's location in the array. Otherwise they were expected to return `null`.

In part (b) students were asked to implement a static method with a 2D array of integers parameter. They were expected to create a 2D array of `Position` references with the same dimensions as the given array. Then they were to use the method they implemented in part (a) to find the position of the successor (integer one greater) for each integer in the given array. They then assigned this successor position to the corresponding element in the new array. Finally, they returned the new array.

In writing the required methods, correct responses demonstrated the ability to search a 2D array, create new `Position` and 2D array objects, return objects and `null`, use parameters and local variables, implement and invoke `static` methods, and demonstrate the principle of code reuse by utilizing a previously implemented method.

Sample: 4A
Score: 8

In part (a) a `boolean` variable `tF` is declared and initialized to `false`. Then all `intArr` elements are compared with `num`. If an equal element is found, a `Position` reference `p` is declared and a `Position` object with the corresponding row and column is constructed and assigned to `p`. Also `tF` is set to `true`. After the loops `!tF` is evaluated to determine if no equal element was found, in which case `null` is returned. Otherwise `p` is returned. This logic earned points 1–4. However, because `p` is declared inside the `if` statement, it is out of scope (inaccessible) in the `return` statement. Therefore point 5 was not earned. Part (a) earned 4 points.

In part (b) the 2D array variable `pos` is created and assigned a correctly sized 2D array of `Position` references, which earned point 1. Nested loops are used to iterate over every element of `intArr`. In the nested loop body, the `findPosition` method is invoked to obtain the `Position` object representing the position of the successor of `intArr[r][c]`. If an `intArr` element does not have a successor, `findPosition` returns `null`. These values are correctly assigned to the corresponding elements in the `pos` 2D array and points 2-4 are earned. Part (b) earned 4 points.

Sample: 4B
Score: 4

In part (a) the extraneous parentheses after each `length` were not penalized, but the second `for` loop header has `intArr[].length`, which does not correctly retrieve the number of columns in `intArr`, and `i++` instead of `j++`. Therefore not every `intArr` element is iterated, so point 1 was not earned. An `intArr` element is compared for equality with `num` in the context of an `intArr` traversal, so point 2 was earned. No object is constructed, so neither point 3 nor point 4 were earned. However, the selected value of `(2,1)` is returned, so point 5 was earned. Part (a) earned 2 points.

Question 4 (continued)

In part (b) point 1 was not earned for several reasons, including the `int[][]` type (Java allows *type name*[][]), `new Array`, and the incorrect number of columns. The invocation of `findPosition` in the context of the `intArr` traversal earned point 3. (Extraneous square brackets when referring to an entire array are not penalized.) Even though point 1 was not earned, the response attempts to declare `Position` as a 2D successor array. Therefore the assignment of the returned `Position` to `Position[i][j]` earned point 2. The loop bounds error for `j` prevented point 4 from being earned. Part (b) earned 2 points.

Sample: 4C
Score: 1

In part (a) the nested loop body consists of an `if-else` statement that always returns during the first iteration. As a result, neither point 1 nor point 4 were earned. Furthermore, both loop bounds are incorrect, and `num` is compared with `int[i][j]` instead of `intArray[i][j]`. These errors result in point 1 and point 2 not being earned. Additionally, `num` is a primitive and primitives don't have methods, so `num.equals(...)` also resulted in point 2 not being earned. No `Position` object is created, so point 3 was not earned. However, `return (i,j)` earned point 5. Part (a) earned 1 point.

In part (b) no 2D array of `Position` objects is created, so point 1 was not earned. Because `intArr` is not traversed, `findPosition` is not called, and no successor positions are determined, points 2–4 were not earned. Part (b) earned no points.