2017

AP CollegeBoard

# AP Computer Science A

## Scoring Guidelines

# 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 1: Digits

| **Part (a)** | `Digits` constructor | **5 points** |
|---|---|---|

**Intent:** *Initialize instance variable using passed parameter*

**+1** Constructs `digitList`

**+1** Identifies a digit in `num`

**+1** Adds at least one identified digit to a list

**+1** Adds all identified digits to a list (*must be in context of a loop)*

**+1** **On exit:** `digitList` contains all and only digits of `num` in the correct order

| **Part (b)** | `isStrictlyIncreasing` | **4 points** |
|---|---|---|

**Intent:** *Determine whether or not elements in `digitList` are in increasing order*

**+1** Compares at least one identified consecutive pair of `digitList` elements

**+1** Determines if a consecutive pair of `digitList` is out of order (*must be in context of a `digitList` traversal*)

**+1** Compares all necessary consecutive pairs of elements (*no bounds errors*)

**+1** Returns `true` iff all consecutive pairs of elements are in order; returns `false` otherwise

**Question-Specific Penalties**

**-2** (q) Uses confused identifier instead of `digitList`

# AP® COMPUTER SCIENCE A
# 2017 SCORING GUIDELINES

## Question 1: Scoring Notes

| **Part (a)** `Digits` constructor | | | **5 points** |
|---|---|---|---|
| Points | Rubric Criteria | Responses earn the point if they ... | Responses will not earn the point if they ... |
| +1 | Constructs `digitList` | | • initialize a local variable instead of `digitList`<br>• create an `ArrayList<int>` |
| +1 | Identifies a digit in `num` | • identify one digit of `num` or a length one substring/character of the `String` representation of `num` | • treat `num` itself as a `String`<br>• convert `num` to a `String` incorrectly |
| +1 | Adds at least one identified digit to a list | • call `add` for some `ArrayList` using the previously identified digit, even if that digit was identified incorrectly | • add `String` or `char` to `digitList` without proper conversion to the correct type |
| +1 | Adds all identified digits to a list (*must be in the context of a loop*) | • call `add` for some `ArrayList` using previously identified digits, even if those digits were identified incorrectly | • identify only 1 digit |
| +1 | **On exit:** `digitList` contains all and only digits of `num` in the correct order | • add to `digitList` even if it is not instantiated properly | • obtain a list with the digits in reverse order<br>• omit one or more digits<br>• add extra digits<br>• mishandle edge case, e.g., 0 or 10<br>• make a bounds error processing the `String` representation of `num` |
| **Part (b)** `isStrictlyIncreasing` | | | **4 points** |
| Points | Rubric Criteria | Responses earn the point if they ... | Responses will not earn the point if they ... |
| +1 | Compares at least one identified consecutive pair of `digitList` elements | • compare two consecutive `Integers` using `compareTo`<br>• explicitly convert two consecutive `Integers` to `ints` and compare those with >=, <= etc.<br>• use auto-unboxing to convert two consecutive `Integers` to `ints` and compare those with >=, <= etc. | • access `digitList` as an array or string<br>• fail to call `.get()`<br>• compare using `!>` |
| +1 | Determines if a consecutive pair of `digitList` is out of order (*must be in context of a `digitList` traversal*) | • determine the correct relationship between the two compared consecutive elements, even if the syntax of the comparison is incorrect | • fail to consider the case where the two elements are equal for the false case |
| +1 | Compares all necessary consecutive pairs of elements (*no bounds errors*) | | • return early |
| +1 | Returns `true` iff all consecutive pairs of elements are in order; returns `false` otherwise | • compare consecutive pairs for inequality, but fail to consider the case when two elements are equal | • return prematurely via `if (...) return false; else return true;` |

# AP® COMPUTER SCIENCE A
# 2017 SCORING GUIDELINES

## Question 1: Digits

*Part (a)*

```
public Digits(int num)
{
      digitList = new ArrayList<Integer>();

      if (num == 0)
      {
            digitList.add(new Integer(0));
      }

      while (num > 0)
      {
            digitList.add(0, new Integer(num % 10));
            num /= 10;
      }
}
```

*Part (b)*

```
public boolean isStrictlyIncreasing()
{
      for (int i = 0; i < digitList.size()-1; i++)
      {
            if (digitList.get(i).intValue() >= digitList.get(i+1).intValue())
            {
                  return false;
            }
      }
      return true;
}
```

**Note:** The solutions shown above were written in compliance with the AP Java subset methods listed for `Integer` objects. Students were allowed to use the automatic "boxing" and "unboxing" of `Integer` objects in their solutions, which eliminates the need to use `"new Integer(...)"` in part (a) and `"intValue()"` in part (b).

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.

# AP® COMPUTER SCIENCE A
# 2017 SCORING GUIDELINES

## Question 2: MultPractice

| **Class:** `MultPractice` | **9 points** |
|---|---|

**Intent:** *Define implementation of class to produce multiplication practice problems*

**+1** Declares header: `public class MultPractice implements StudyPractice`

**+1** Declares all necessary `private` instance variables

**+2** Constructor

- **+1** Declares header: `public MultPractice(int __, int __)`
- **+1** Initializes all instance variables using parameters

**+3** `getProblem` method

- **+1** Declares header: `public String getProblem()`
- **+1** Builds string with current values of instance variables
- **+1** Returns constructed string

**+2** `nextProblem` method

- **+1** Declares header: `public void nextProblem()`
- **+1** Updates instance variable(s) to reflect incremented second number

# AP® COMPUTER SCIENCE A
# 2017 SCORING GUIDELINES

## Question 2: Scoring Notes

| **Class** `MultPractice` | | | **9 points** |
|---|---|---|---|
| Points | Rubric Criteria | Responses earn the point if they ... | Responses will not earn the point if they ... |
| +1 | Declares header: `public class MultPractice implements StudyPractice` | • omit keyword `public` | • declare class `private` |
| +1 | Declares all necessary `private` instance variables | • declare the unchanging instance variable as `final` | • declare variables as `static`<br>• omit keyword `private` |
| +2 | Constructor | | |
| +1 | Declares header: `public MultPractice (int ___, int ___)` | • omit keyword `public` | |
| +1 | Initializes all instance variables using parameters | | • fail to declare nonlocal variables<br>• initialize local variables instead of instance variables<br>• assign variables to parameters |
| +3 | `getProblem` method | | |
| +1 | Declares header: `public String getProblem()` | | • fail to declare method `public` |
| +1 | Builds string with current values of instance variables | • write appropriate code in a method other than `getProblem`<br>• make capitalization or spacing errors | • fail to declare nonlocal variables<br>• fail to use instance variables<br>• miscast `(String) intVar`<br>• call `intVar.toString()` |
| +1 | Returns constructed string | | • return a literal string |
| +2 | `nextProblem` method | | |
| +1 | Declares header: `public void nextProblem()` | | • fail to declare method `public` |
| +1 | Updates instance variable(s) to reflect incremented second number | | • fail to declare non-local variables |

# AP® COMPUTER SCIENCE A
# 2017 SCORING GUIDELINES

## Question 2: MultPractice

```
public class MultPractice implements StudyPractice
{
      private int first;
      private int second;

      public MultPractice(int num1, int num2)
      {
            first = num1;
            second = num2;
      }

      public String getProblem()
      {
            return first + " TIMES " + second;
      }

      public void nextProblem()
      {
            second++;
      }
}
```

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.

# 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);`<br>• 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);`<br>• 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);`<br>• compute the correct value to be returned in all cases, but no return statement exists for any case | • shorten string being searched<br>• 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<br>• 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.

# 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<br>• confuse row and column bounds<br>• 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`<br>• use `(r,c)` instead of `Position(r,c)` |
| +1 | Selects constructed object when `intArr` element identified; `null` when not | • use `"null"` instead of `null`<br>• construct a `String` object using row and column indices | • use `if (...) return; else return null;` inside loop<br>• 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`<br>• call `findPosition` with a single argument<br>• 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`<br>• 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<br>• only assign non-null entries to `SuccessorArray` | • reimplement the code from `findPosition` but mishandle the null case.<br>• 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.