

2021

AP[®]

 CollegeBoard

AP[®] Computer Science A

Scoring Guidelines

© 2021 College Board. College Board, Advanced Placement, AP, AP Central, and the acorn logo are registered trademarks of College Board. Visit College Board on the web: collegeboard.org.

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

Applying the Scoring Criteria

Apply the question scoring criteria 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.*

Question 1: Methods and Control Structures**9 points****Canonical solution**

- (a) `public int scoreGuess(String guess)` **5 points**
- ```
{
 int count = 0;

 for (int i = 0; i <= secret.length() - guess.length(); i++)
 {
 if (secret.substring(i, i + guess.length()).equals(guess))
 {
 count++;
 }
 }

 return count * guess.length() * guess.length();
}
```
- (b) `public String findBetterGuess(String guess1, String guess2)` **4 points**
- ```
{
    if (scoreGuess(guess1) > scoreGuess(guess2))
    {
        return guess1;
    }
    if (scoreGuess(guess2) > scoreGuess(guess1))
    {
        return guess2;
    }
    if (guess1.compareTo(guess2) > 0)
    {
        return guess1;
    }
    return guess2;
}
```

(a) `scoreGuess`

Scoring Criteria		Decision Rules	
1	Compares <code>guess</code> to a substring of <code>secret</code>	Responses can still earn the point even if they only call <code>secret.indexOf(guess)</code> Responses will not earn the point if they use <code>==</code> instead of <code>equals</code>	1 point
2	Uses a substring of <code>secret</code> with correct length for comparison with <code>guess</code>	Responses can still earn the point even if they - only call <code>secret.indexOf(guess)</code> - use <code>==</code> instead of <code>equals</code>	1 point
3	Loops through all necessary substrings of <code>secret</code> (<i>no bounds errors</i>)	Responses will not earn the point if they skip overlapping occurrences	1 point
4	Counts number of identified occurrences of <code>guess</code> within <code>secret</code> (<i>in the context of a condition involving both <code>secret</code> and <code>guess</code></i>)	Responses can still earn the point even if they - initialize count incorrectly or not at all - identify occurrences incorrectly	1 point
5	Calculates and returns correct final score (<i>algorithm</i>)	Responses will not earn the point if they - initialize count incorrectly or not at all - fail to use a loop - fail to compare <code>guess</code> to multiple substrings of <code>secret</code> - count the same matching substring more than once - use a changed or incorrect <code>guess</code> length when computing the score	1 point
Total for part (a)			5 points

(b) `findBetterGuess`

Scoring Criteria		Decision Rules	
6	Calls <code>scoreGuess</code> to get scores for <code>guess1</code> and <code>guess2</code>	Responses will not earn the point if they - fail to include parameters in the method calls - call the method on an object or class other than <code>this</code>	1 point
7	Compares the scores	Responses will not earn the point if they - only compare using <code>==</code> or <code>!=</code> - fail to use the result of the comparison in a conditional statement	1 point
8	Determines which of <code>guess1</code> and <code>guess2</code> is alphabetically greater	Responses can still earn the point even if they reverse the comparison Responses will not earn the point if they - reimplement <code>compareTo</code> incorrectly - use result of <code>compareTo</code> as if <code>boolean</code>	1 point
9	Returns the identified <code>guess1</code> or <code>guess2</code> (<i>algorithm</i>)	Responses can still earn the point even if they - call <code>scoreGuess</code> incorrectly - compare strings incorrectly Responses will not earn the point if they - reverse a comparison - omit either comparison - fail to return a guess in some case	1 point
Total for part (b)			4 points
Question-specific penalties			
None			
Total for question 1			9 points

Applying the Scoring Criteria

Apply the question scoring criteria 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.*

Question 2: Class Design**9 points****Canonical solution**

```
public class CombinedTable
{
    private SingleTable table1;
    private SingleTable table2;

    public CombinedTable(SingleTable tab1, SingleTable tab2)
    {
        table1 = tab1;
        table2 = tab2;
    }

    public boolean canSeat(int n)
    {
        if (table1.getNumSeats() + table2.getNumSeats() - 2 >= n)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public double getDesirability()
    {
        if (table1.getHeight() == table2.getHeight())
        {
            return (table1.getViewQuality() +
                    table2.getViewQuality()) / 2;
        }
        else
        {
            return ((table1.getViewQuality() +
                    table2.getViewQuality()) / 2) - 10;
        }
    }
}
```

9 points

CombinedTable

Scoring Criteria	Decision Rules	
1 Declares class header: <code>class CombinedTable</code> and constructor header: <code>CombinedTable(SingleTable ____, SingleTable ____)</code> <i>(must not be private)</i>	Responses can still earn the point even if they declare the class header as <code>class CombinedTable extends SingleTable</code>	1 point
2 Declares appropriate <code>private</code> instance variables including at least two <code>SingleTable</code> references	Responses can still earn the point even if they declare an additional instance variable to cache the number of seats at the combined table Responses will not earn the point if they • declare and initialize local variables in the constructor instead of instance variables • declare additional instance variable(s) that cache the desirability rating • omit keyword <code>private</code> • declare variables outside the class	1 point
3 Constructor initializes instance variables using parameters	Responses can still earn the point even if they declare and initialize local variables in the constructor instead of instance variables	1 point
4 Declares header: <code>public boolean canSeat(int __)</code>		1 point
5 Calls <code>getNumSeats</code> on a <code>SingleTable</code> object	Responses can still earn the point even if they call <code>getNumSeats</code> on constructor parameters or local variables of type <code>SingleTable</code> in the constructor Responses will not earn the point if they call the <code>SingleTable</code> accessor method on something other than a <code>SingleTable</code> object	1 point
6 <code>canSeat(n)</code> returns <code>true</code> if and only if sum of seats of two tables $- 2 \geq n$	Responses can still earn the point even if they call <code>getNumSeats</code> incorrectly	1 point
7 Declares header: <code>public double getDesirability()</code>		1 point
8 Calls <code>getHeight</code> and <code>getViewQuality</code> on <code>SingleTable</code> objects	Responses can still earn the point even if they call <code>getHeight</code> or <code>getViewQuality</code> on constructor parameters or local variables of type <code>SingleTable</code> in the constructor	1 point

		Responses will not earn the point if they call the <code>SingleTable</code> accessor methods on something other than a <code>SingleTable</code> object	
9	<code>getDesirability</code> computes average of constituent tables' view desirabilities	Responses can still earn the point even if they - call <code>getHeight</code> or <code>getViewQuality</code> on constructor parameters or local variables of type <code>SingleTable</code> in the constructor - fail to return the computed average (<i>return is not assessed</i>) Responses will not earn the point if they - fail to have an <code>if</code> statement and a correct calculation - choose the incorrect value (average vs. average – 10) based on evaluation of the <code>if</code> statement condition	1 point
Question-specific penalties			
None			
Total for question 2			9 points

Applying the Scoring Criteria

Apply the question scoring criteria 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.*

Question 3: Array / ArrayList**9 points****Canonical solution****(a)**

```
public void addMembers(String[] names, int gradYear)
{
    for (String n : names)
    {
        MemberInfo newM = new MemberInfo(n, gradYear, true);
        memberList.add(newM);
    }
}
```

3 points**(b)**

```
public ArrayList<MemberInfo> removeMembers(int year)
{
    ArrayList<MemberInfo> removed = new ArrayList<MemberInfo>();

    for (int i = memberList.size() - 1; i >= 0; i--)
    {
        if (memberList.get(i).getGradYear() <= year)
        {
            if (memberList.get(i).inGoodStanding())
            {
                removed.add(memberList.get(i));
            }
            memberList.remove(i);
        }
    }
    return removed;
}
```

6 points

(a) `addMembers`

Scoring Criteria		Decision Rules	
1	Accesses all elements of <code>names</code> (<i>no bounds errors</i>)	Responses will not earn the point if they fail to access elements of the array, even if loop bounds are correct	1 point
2	Instantiates a <code>MemberInfo</code> object with name from array, provided year, and good standing		1 point
3	Adds <code>MemberInfo</code> objects to <code>memberList</code> (<i>in the context of a loop</i>)	Responses can earn the point even if they instantiate <code>MemberInfo</code> objects incorrectly	1 point
Total for part (a)			3 points

(b) `removeMembers`

Scoring Criteria		Decision Rules	
4	Declares and initializes an <code>ArrayList</code> of <code>MemberInfo</code> objects	Responses will not earn the point if they initialize the variable with a reference to the instance variable	1 point
5	Accesses all elements of <code>memberList</code> for potential removal (<i>no bounds errors</i>)	Responses will not earn the point if they - fail to use <code>get(i)</code> - fail to attempt to remove an element - skip an element - throw an exception due to removing	1 point
6	Calls <code>getGradYear</code> or <code>inGoodStanding</code>	Responses can still earn the point even if they call only one of the methods Responses will not earn the point if they - ever include parameters in either method call - ever call either method on an object other than <code>MemberInfo</code>	1 point
7	Distinguishes any three cases, based on graduation status and standing	Responses will not earn the point if they fail to behave differently in all three cases	1 point
8	Identifies graduating members	Responses can still earn the point even if they - fail to distinguish three cases - fail to access standing at all - access the graduating year incorrectly Responses will not earn the point if they confuse <code><</code> and <code><=</code> in the comparison	1 point
9	Removes appropriate members from <code>memberList</code> and adds appropriate members to the <code>ArrayList</code> to be returned	Responses can still earn the point even if they - call <code>getGradYear</code> or <code>inGoodStanding</code> incorrectly - access elements of <code>memberList</code> incorrectly - initialize the <code>ArrayList</code> incorrectly - fail to return the list that was built (<i>return is not assessed</i>) Responses will not earn the point if they - fail to declare an <code>ArrayList</code> to return - fail to distinguish the correct three cases, with the exception of confusing the <code><</code> and <code><=</code> in the comparison	1 point
Total for part (b)			6 points

Question-specific penalties	
None	
Total for question 3 9 points	

Applying the Scoring Criteria

Apply the question scoring criteria 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.*

Question 4: 2D Array**9 points****Canonical solution****(a)**

```
public static boolean isNonZeroRow(int[][] array2D, int r)
{
    for (int col = 0; col < array2D[0].length; col++)
    {
        if (array2D[r][col] == 0)
        {
            return false;
        }
    }
    return true;
}
```

3 points**(b)**

```
public static int[][] resize(int[][] array2D)
{
    int numRows = array2D.length;
    int numCols = array2D[0].length;

    int[][] result = new int[numNonZeroRows(array2D)][numCols];
    int newRowIndex = 0;

    for (int r = 0; r < numRows; r++)
    {
        if (isNonZeroRow(array2D, r))
        {
            for (int c = 0; c < numCols; c++)
            {
                result[newRowIndex][c] = array2D[r][c];
            }
            newRowIndex++;
        }
    }
    return result;
}
```

6 points

(a) `isNonZero`

Scoring Criteria		Decision Rules	
1	Compares an item from <code>array2D</code> with <code>0</code>	Responses will not earn the point if they fail to attempt the comparison, even if they access an item from <code>array2D</code>	1 point
2	Accesses every item from row <code>r</code> of 2D array (<i>no bounds errors</i>)	Responses can still earn the point even if they return early from an otherwise correctly-bounded loop	1 point
3	Returns <code>true</code> if and only if row contains no zeros	Responses can still earn the point even if they process a column of the 2D array rather than a row Responses will not earn the point if they fail to return a value in some cases	1 point
Total for part (a)			3 points

(b) `resize`

Scoring Criteria		Decision Rules	
4	Calls <code>numNonZeroRows</code> and <code>isNonZeroRow</code>	Responses can still earn the point even if they fail to use or store the return value Responses will not earn the point if they - include incorrect number or type of parameters - call methods on an object or class other than <code>ArrayResizer</code>	1 point
5	Identifies rows with no zeros (<i>in the context of an if</i>)	Responses can still earn the point even if they call <code>isNonZeroRow</code> incorrectly, if the row being tested is clearly identified (index or reference)	1 point
6	Declares and creates a new 2D array of the correct size	Response will not earn the point if they transpose the dimensions of the created array	1 point
7	Maintains an index in the new array	Responses will not earn the point if they - fail to declare, initialize, and update a different index - maintain the index in a way that overwrites, skips, or duplicates rows	1 point
8	Traverses all necessary elements of <code>array2D</code> (<i>no bounds errors</i>)	Responses can still earn the point even if they - cause a bounds error by declaring and creating a new 2D array of an incorrect size - fail to maintain an index in the new array correctly, resulting in a bounds error - fail to access individual elements in a nested loop, if they access each row as an entire row Responses will not earn the point if they transpose coordinates, leading to a bounds error and/or copying columns	1 point
9	Copies all and only rows identified as having no zero elements into the new array	Responses can still earn the point even if they - copy a reference - identify rows incorrectly, if the logical sense can be determined and is correct - copy columns instead of rows, consistent with the dimensions of the created 2D array	1 point

Responses **will not** earn the point if they

- remove or overwrite data from `array2D` (instead of or in addition to copying it to the new array)
- reverse the logical sense of which rows to copy

Total for part (b) 6 points

Question-specific penalties

-1 (u) Use `array2D[].length` to refer to the number of columns in a row of the 2D array

Total for question 4 9 points