AP Computer Science Multiple-Choice Review Tips

1. Be familiar with the format of the AP CS A multiple-choice section.

You will have 1 hour and 30 minutes to answer 40 multiple-choice questions. Keep in mind that this sections counts for half of your overall exam score. You will be tested on your ability to predict how code segments will work, analyze object-oriented program designs, modify and correct segments of code, and more.

2. Know the concepts covered on the exam.

According to the College Board, multiple-choice exam questions can be classified according to certain categories. It's important that you look over this table to see which questions are tested the most. For example, you should spend more time studying programming fundamentals and less time studying software engineering.

Classification Category	Percentage of Multiple-Choice Items
Programming Fundamentals	**55-75%**
Data Structures	**20-40%**
Logic	**5-15%**
Algorithms/Problem Solving	**24-45%**
Object-Oriented Programming	**15-25%**
Recursion	**5-15%**
Software Engineering	**2-10%**

Note: Total percentage is greater than 100% because some questions can be classified in more than one category.

3. Employ standard multiple-choice test-taking strategies.

The multiple-choice section of the AP Computer Science A exam can be overwhelming. You have limited time and 40 complex questions to analyze and answer. Sometimes, the best thing to do is take a deep breath and remember some of the essential multiple-choice test taking strategies. You're probably aware of some of these, but it's still worth reminding you.

– Answer EVERY question, even if it's just a complete guess. You will not be docked for providing a wrong answer. If you had the time to read the question, you have time to make an educated guess.
– Read all of the answer choices before making your final decision.
– Use the process of elimination to narrow down your choices. If you can narrow it down to two choices, you have a 50% chance of choosing the right answer, as opposed to a 25% chance.
– Keep track of the time. Make sure you glance at your watch every so often to make sure you're not running out of time.
– Remember that a machine scores this section of the exam, so make sure you're filling in the answer choices fully and darkly.
– Use the available space for any necessary scratch work. Keep in mind, however, that no credit will given for anything written in the exam booklet.
– Feel free to mark up the question. Circle or underline if it helps.
– Depending on your test-taking style, reading the question before even looking at the possible answer choices may work best for you. Conversely, looking at the answer choices and then reading the question may work best. Find a style that works for you.

4. Know what a typical multiple-choice question looks like.

Practicing by taking multiple practice exams over the course of the year is the best way to become comfortable and confident with this section of the exam. The majority of questions will include a snippet of code, which you will have to analyze in one way or another. Sometimes you will given an output and have to choose the code segment that produces that output. Other times, you may be asked to predict what a particular code segment will print out. More complex questions might ask you to change segments of code to perform a specific task, evaluate recursion methods, describe what will happen when a code segment is executed, and analyze code that may have many correct answers. Take a look at the sample multiple-choice questions offered by the College Board in its AP Computer Science A Course Description to get a feel for the complexity of the questions you might encounter on the exam.

AP Computer Science Free-Response Tips

1. Avoid the most common errors.

In CS, you will make mistakes. It comes with the territory. However, as long as you are aware of the most common mistakes and errors, you can hopefully avoid them on the free-response section of the exam. The most common errors include:

– Being off by one in loops
– Not initializing a variable
– Dangling else
– Failure to return a required value
– Using the wrong identifier
– Not returning a statement in a non-void method
– Not satisfying the required post-conditions
– Not including the word public or private when required
– Modifying a constant
– Using == when comparing two objects
– Putting double quotes around “null” instead of just null
– Using local variables but not declaring them
– Missing { }, (), or semicolons
– Double quoting “true” and “false” when using them as Boolean values
– Including extraneous code that causes side effects
– Not checking for boundary cases
– Forgetting the word new in a constructor call statement
– Confusing = with ==
– Confusing [] with ()
– Assigning values incorrectly (putting x + 2 = y, instead of y = x + 2)
– Using a class name instead of a variable name

2. Write legibly, clearly, and with good programming style.

t’s always important to remember that humans will be grading the free-response section of the exam. This means you should aim for clarity in your programming responses. You want your responses to be as easy to grade as possible. Keep in mind the following when writing your programming responses:

– Assign meaningful variable names
– Write neatly and legibly
– Always indent properly (this is a big one because if you accidentally miss a curly brace or semicolon, proper indentation will save you)
– Position curly braces clearly

– Keep it organized
– Use white space liberally
– Do not write too small or too large
– Put each statement on a separate line

3. Do not use specific numbers, strings, or dimensions of arrays in your code.

You want to make your program work with any potential number, string, or dimension, not just the one given to you as an example in the question. This is a common mistake that many AP CS students make. For example, if a question is about a two-dimensional array and it shows you an image of a 3 by 4 array, do not use the numbers 3 and 4 in your code. Instead, depending on what the question is asking, you'd want to make your code work with an array of any size. Remember, do notuse specific numbers!

4. Attempt every part of the question.

Most AP Computer Science free-response questions will include multiple parts (part a, part b, part c, etc.). It may surprise you that part (a) is usually harder than parts (b) and (c). The problem with this is that many test-takers get discouraged when they don't know how to answer part (a), so they don't even attempt the other parts. Don't do this! Each part is graded independently from the previous parts. This means that if you leave part (a) blank or provide an incorrect answer, you could still get full or partial credit for the other parts! However, do not re-implement code from earlier parts in later parts, even if that particular code is 100% correct. This is a waste of time and you could potentially lose points.

5. Make sure you've actually answered the question.

Sounds like a no-brainer, right? Not so much. Some students get so caught up in their answer that they forget to make sure they're actually addressing the problem or answering the question. To avoid this, underline, circle, or star important details when you first read the problem. Reread the question. Look for phrases like "You will receive no credit if…". When you've finished writing your code, look at the problem again. Look at the information you underlined or circled. Did you actually answer the question?

6. Always write down some code in your answer.

You will be given no credit for simply describing what you would do if you had the time to write out the code (this is not an English exam). You could receive some credit, however, for attempting some aspect of the code. You may not know how to write the full code to address the problem, but always write at least some code for the parts you do Partial credit may be given for having the correct loop bounds, attempting to sum values in an array, etc.

7. Remember that elegance of code does not matter.

For you advanced programmers out there, elegant coding does not count on the AP CS exam. Don't try and show off, no matter how much you want to. Stick to what you know. You could potentially waste time trying to write really simple, efficient code, or even trick yourself in the process. A brute-force approach is best on the exam

8. Stay within the AP Java subset.

The subset if given to you for a reason. Even if you went outside the subset in your AP CS class or if you're a Java pro, don't get fancy on the exam. Keep it simple and use the AP Java subset.

9. Follow Java naming conventions.

A simple tip, but make sure you're naming your methods, variables, and parameters correctly. Start names with a lowercase letter and keep names meaningful but not too verbose. Use your best judgment. For example, "count" would be better than "a," but " k" would be better than "loopControlVariable." Also, make sure you look at the question carefully before naming things in your code. Sometimes the question will contain names that you should use in your code.

10. Don't waste time including comments.

Comments are completely disregarded by AP readers. You will therefore receive no credit for them. Your code should never be so confusing that you have to include comments to show the AP reader what you're doing. The only time you should include comments is if it helps you organize your thoughts and stay on track. Since you'll be under a time crunch on the AP Computer Science FRQ, don't waste your valuable time on comments.

11. Look out for hints in the question.

Sometimes an FRQ will suggest that you use a particular algorithm (it might say something like "you may use the following algorithm"). They're not just throwing that information out there for no reason!

12. Code according to the specifications and preconditions and postconditions.

Don't code anything outside of the specifications! You don't want to include any "bells and whistles," since you are likely to lose points if you do. It's important to note that you should never add System.out.print unless you are asked to. Additionally, don't include any unnecessary checks to your code.

APPENDIX B

Exam Appendix — Java Quick Reference

Accessible methods from the Java library that may be included on the exam

class java.lang.Object

- `boolean equals(Object other)`
- `String toString()`

class java.lang.Integer

- `Integer(int value)`
- `int intValue()`
- `Integer.MIN_VALUE` // minimum value represented by an `int` or `Integer`
- `Integer.MAX_VALUE` // maximum value represented by an `int` or `Integer`

class java.lang.Double

- `Double(double value)`
- `double doubleValue()`

class java.lang.String

- `int length()`
- `String substring(int from, int to)` // returns the substring beginning at `from`
  // and ending at `to-1`
- `String substring(int from)` // returns `substring(from, length())`
- `int indexOf(String str)` // returns the index of the first occurrence of `str`;
  // returns `-1` if not found
- `int compareTo(String other)` // returns a value < 0 if `this` is less than `other`
  // returns a value = 0 if `this` is equal to `other`
  // returns a value > 0 if `this` is greater than `other`

class java.lang.Math

- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`
- `static double random()` // returns a `double` in the range [0.0, 1.0)

interface java.util.List<E>

- `int size()`
- `boolean add(E obj)` // appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` // inserts `obj` at position `index` ($0 \le \texttt{index} \le \texttt{size}$),
  // moving elements at position `index` and higher
  // to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` // replaces the element at position `index` with `obj`
  // returns the element formerly at the specified position
- `E remove(int index)` // removes element from position `index`, moving elements
  // at position `index + 1` and higher to the left
  // (subtracts 1 from their indices) and adjusts size
  // returns the element formerly at the specified position

class java.util.ArrayList<E> implements java.util.List<E>