AP® COMPUTER SCIENCE A
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

(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)

Mr Lee's 1-Point Penalty:

- Inefficient, "long winded" or "messy" difficult to understand code which takes longer to write than standard more efficient solutions.
    - In an exam you need to save time by writing quickly hand writable efficient code which is easy for AP readers to understand.

No Penalty

- Extraneous code with no side effect (e.g., precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- Keyword used as an identifier
- Common mathematical symbols used for operators (x • ÷ ≤ ≥< > ≠)
- `[ ]` vs. `()`
- Extraneous [ ] when referencing entire array
- `[i,j]` instead of `[i] [j]`
- `=` instead of `==` and vice versa
- Missing `{}` where indentation clearly conveys intent
- 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, "total" instead of "totl". As a counterexample, that if the code declares "int G=99 , g=O; ", 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.*

1D Arrays – TokenPass FRQ

A multiplayer game called Token Pass has the following rules. Each player begins with a random number of tokens (at least 1, but no more than 10) that are placed on a linear game board. There is one position on the game board for each player. After the game board has been filled, a player is randomly chosen to begin the game. Each position on the board is numbered, starting with 0.

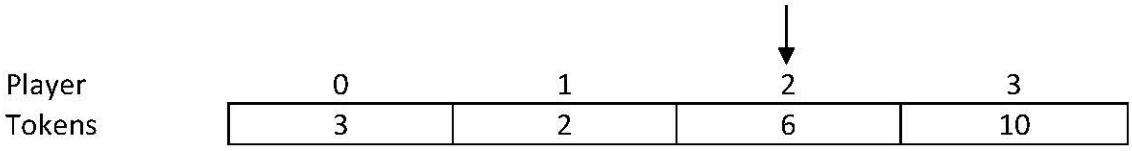
The following rules apply for a player's turn.

- The tokens are collected and removed from the game board at that player's position.
- The collected tokens are distributed one at a time, to each player, beginning with the next player in order of increasing position.
- If there are still tokens to distribute after the player at the highest position gets a token, the next token will be distributed to the player at position 0.
- The distribution of tokens continues until there are no more tokens to distribute.

The Token Pass game board is represented by an array of integers. The indexes of the array represent the player positions on the game board, and the corresponding values in the array represent the number of tokens that each player has. The following example illustrates one player's turn.

Example

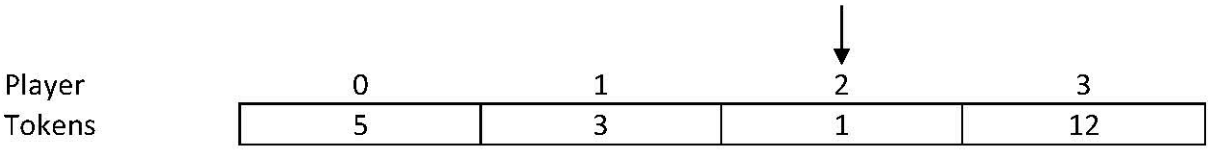
The following represents a game with 4 players. The player at position 2 was chosen to go first.



The tokens at position 2 are collected and distributed as follows.

- 1st token - to position 3 (The highest position is reached, so the next token goes to position 0.)
- 2nd token - to position 0
- 3rd token - to position 1
- 4th token - to position 2
- 5th token - to position 3 (The highest position is reached, so the next token goes to position 0.)
- 6th token - to position 0

After player 2's turn, the values in the array will be as follows.




```
int[] board;
int  currentPlayer;
```

1D Arrays – TokenPass FRQ

(a) Write a code segment for the `TokenPass` game. The parameter `playerCount` represents the number of players in the game. The code segment should create the `board` array to contain `playerCount` elements and fill the array with random numbers between `1` and `10`, inclusive. The code segment should also initialize the variable `currentPlayer` to a random number between `0` and `playerCount - 1`, inclusive.

Complete the code segment below.

```
/** Creates the board array to be of size playerCount and
 *  fills it with random integer values from 1 to 10,
 *  inclusive. Initializes currentPlayer to a random integer
 *  value in the range between 0 and playerCount - 1,
 *  inclusive.
 *  @param playerCount the number of players
 */
int playerCount
```

(b) Write another code segment.

The tokens are collected and removed from the game board at the current player's position. These tokens are distributed, one at a time, to each player, beginning with the next higher position, until there are no more tokens to distribute.

Information repeated from the beginning of the question.

```
int[] board;
int currentPlayer;
int playercount;
```

Complete the code segment below.

```
/** Distributes the tokens from the current player's position
 *  one at a time to each player in the game. Distribution
 *  begins with the next position and continues until all the
 *  tokens have been distributed. If there are still tokens to
 *  distribute when the player at the highest position is
 *  reached, the next token will be distributed to the player
 *  at position 0.
 *  Precondition: the current player has at least one token.
 *  Postcondition: the current player has not changed.
 */
```