1D Arrays – Robot FRQ

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 – Robot FRQ

The PR2004 is a robot that automatically gathers toys and other items scattered in a tiled hallway. A tiled hallway has a wall at each end and consists of a single row of tiles, each with some number of items to be gathered.

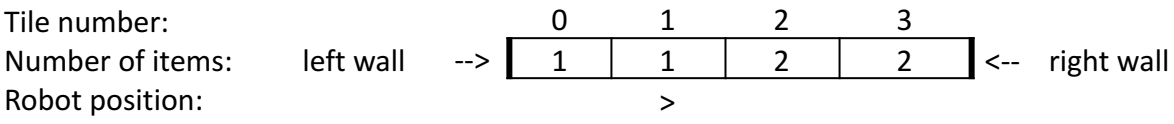
The PR2004 robot is initialized with a starting position and an array that contains the number of items on each tile. Initially the robot is facing right, meaning that it is facing toward higher-numbered tiles.

The PR2004 robot makes a sequence of moves until there are no items remaining on any tile. A move is defined as follows.

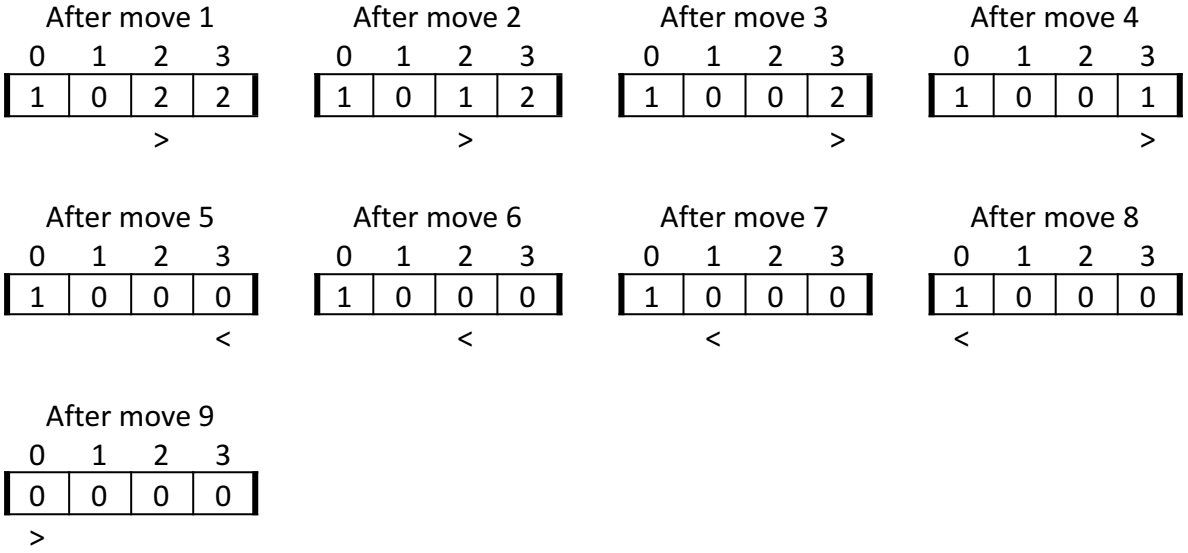
1. If there are any items on the current tile, then one item is removed.
2. If there are more items on the current tile, then the robot remains on the current tile facing the same direction.
3. If there are no more items on the current tile

a) if the robot can move forward, it advances to the next tile in the direction that it is facing;
b) otherwise, if the robot cannot move forward, it reverses direction and does not change position.

In the following example, the position and direction of the robot are indicated by "<" or">" and the entries in the diagram indicate the number of items to be gathered on each tile. There are four tiles in this hallway. The starting state of the robot is illustrated in the following diagram.



The following sequence shows the configuration of the hallway and the robot after each move.



After nine moves, the robot stops because the hall is clear.

```
int[] hall ; // precondition: an array of valid ints >= 0

int pos;     // current position (tile number) of Robot
             // precondition: >= 0 && < hall.length

boolean facingRight; // true means this Robot is facing right
                     // precondition: a valid boolean
```

The number of items on each tile in the hall is stored in the corresponding entry in the array *hall*. The current position is stored in the variable *pos*. The *boolean* variable facingRight is *true* if the Robot is facing to the right and is *false* otherwise.

Write a code segment which prints *true* if the robot has a wall immediately in front of it, so that it cannot move forward. Otherwise, it prints *false*.

Complete code segment below.

```
// postcondition: prints true if this Robot has a wall
//                immediately in front of it, so that it cannot
//                move forward; otherwise, prints false
```