



## AP<sup>®</sup> Computer Science A 2004 Sample Student Responses

**The materials included in these files are intended for noncommercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program<sup>®</sup>. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. This permission does not apply to any third-party copyrights contained herein. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here.**

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,500 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT<sup>®</sup>, the PSAT/NMSQT<sup>®</sup>, and the Advanced Placement Program<sup>®</sup> (AP<sup>®</sup>). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit [www.collegeboard.com](http://www.collegeboard.com)

Copyright © 2004 College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, AP Vertical Teams, APCD, Pacesetter, Pre-AP, SAT, Student Search Service, and the acorn logo are registered trademarks of the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service. Other products and services may be trademarks of their respective owners.

For the College Board's online home for AP professionals, visit AP Central at [apcentral.collegeboard.com](http://apcentral.collegeboard.com).

- (a) Write the Robot method `forwardMoveBlocked`. Method `forwardMoveBlocked` returns `true` if the robot has a wall immediately in front of it, so that it cannot move forward. Otherwise, `forwardMoveBlocked` returns `false`.

Complete method `forwardMoveBlocked` below.

```
// postcondition: returns true if this Robot has a wall immediately in
//                 front of it, so that it cannot move forward;
//                 otherwise, returns false
private boolean forwardMoveBlocked()
{
    if ( facingRight && pos == hall.length - 1)
        return true;
    else if (!facingRight && pos == 0)
        return true;
    else
        return false;
}
```

GO ON TO THE NEXT PAGE.

- (b) Write the Robot method `move`. Method `move` has the robot carry out one move as specified at the beginning of the question. The specification for a move is repeated here for your convenience.
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 writing `move`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `move` below.

```
// postcondition: one move has been made according to the
//                specifications above and the state of this
//                Robot has been updated
private void move()
{
    if (hall[pos] > 0)
        hall[pos] -= 1;
    if (hall[pos] == 0 && forwardMoveBlocked() != true)
    {
        if (facingRight)
            pos -= 1;
        else
            pos += 1;
    }
    else if (hall[pos] == 0 && forwardMoveBlocked() == true)
    {
        if (facingRight)
            facingRight = false;
        else
            facingRight = true;
    }
}
```

Part (c) begins on page 20.

GO ON TO THE NEXT PAGE.

- (c) Write the Robot method `clearHall`. Method `clearHall` clears the hallway, repeatedly having this robot make a move until the hallway has no items, and returns the number of moves made.

In the example at the beginning of this problem, `clearHall` would take the robot through the moves shown and return 9, leaving the robot in the state shown in the final diagram.

In writing `clearHall`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `clearHall` below.

```
// postcondition: no more items remain in the hallway;
//               returns the number of moves made
public int clearHall()
{
    int numMoves = 0,
    while ( hallIsClear () != true)
    {
        move ();
        numMoves ++ ;
    }
    return numMoves ;
}
```

- (a) Write the Robot method `forwardMoveBlocked`. Method `forwardMoveBlocked` returns `true` if the robot has a wall immediately in front of it, so that it cannot move forward. Otherwise, `forwardMoveBlocked` returns `false`.

Complete method `forwardMoveBlocked` below.

```
// postcondition: returns true if this Robot has a wall immediately in
//                 front of it, so that it cannot move forward;
//                 otherwise, returns false
private boolean forwardMoveBlocked()
{
    if (facingRight) {
        if pos == hall.length()
            return true;
        else return false;
    }
    else if pos == 0
        return true;
    return false;
}
```

**GO ON TO THE NEXT PAGE.**

- (b) Write the Robot method move. Method move has the robot carry out one move as specified at the beginning of the question. The specification for a move is repeated here for your convenience.
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 writing move, you may use any of the other methods in the Robot class. Assume these methods work as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method move below.

```
// postcondition: one move has been made according to the
//                specifications above and the state of this
//                Robot has been updated
```

```
private void move()
{
    if (hall[pos] > 0)
        hall[pos]--;
    if (hall[pos] == 0)
        if (forwardMoveBlocked())
            !facing Right;
        else
            if (facing Right)
                pos++;
            else pos--;
    }
}
```

Part (c) begins on page 20.

**GO ON TO THE NEXT PAGE.**

- (c) Write the Robot method `clearHall`. Method `clearHall` clears the hallway, repeatedly having this robot make a move until the hallway has no items, and returns the number of moves made.

In the example at the beginning of this problem, `clearHall` would take the robot through the moves shown and return 9, leaving the robot in the state shown in the final diagram.

In writing `clearHall`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `clearHall` below.

```
// postcondition: no more items remain in the hallway;
//               returns the number of moves made
public int clearHall()
{
    int count = 0;
    int numofToys = 0;
    for (int i = 0; i < hall.length(); i++)
        numofToys += hall[i];
    while (numofToys > 0) {
        move();
        count++;
        numofToys = 0;
        for (int i = 0; i < hall.length(); i++)
            numofToys += hall[i];
    }
    return count;
}
```

A4 C1

- (a) Write the Robot method `forwardMoveBlocked`. Method `forwardMoveBlocked` returns `true` if the robot has a wall immediately in front of it, so that it cannot move forward. Otherwise, `forwardMoveBlocked` returns `false`.

Complete method `forwardMoveBlocked` below.

```
// postcondition: returns true if this Robot has a wall immediately in
//                front of it, so that it cannot move forward;
//                otherwise, returns false
private boolean forwardMoveBlocked()
{
```

```
    if (hall.length - 1 == pos && facingRight)
        return true;
    else if (pos == 0 && !facingRight)
        return true;
    else
        return false;
}
```

GO ON TO THE NEXT PAGE.



C<sub>2</sub>

(b) Write the `Robot` method `move`. Method `move` has the robot carry out one move as specified at the beginning of the question. The specification for a move is repeated here for your convenience.

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 writing `move`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `move` below.

```
// postcondition: one move has been made according to the
//                specifications above and the state of this
//                Robot has been updated
private void move()
{
```

```
    for (int k = 0, k < hall length, k++)
    {
        int p = hall[k];
        while (p > 0) {
            p--;
        }
    }
}
```

Part (c) begins on page 20.

**GO ON TO THE NEXT PAGE.**

- (c) Write the Robot method `clearHall`. Method `clearHall` clears the hallway, repeatedly having this robot make a move until the hallway has no items, and returns the number of moves made.

In the example at the beginning of this problem, `clearHall` would take the robot through the moves shown and return 9, leaving the robot in the state shown in the final diagram.

In writing `clearHall`, you may use any of the other methods in the `Robot` class. Assume these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement the functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `clearHall` below.

```
// postcondition: no more items remain in the hallway;
//               returns the number of moves made
public int clearHall()
{
    int count = 0;
    for (int k = 0; k < hall.length; k++)
    {
        while (hall[k] != 0)
        {
            move(k);
            count++;
        }
    }
    return count;
}
```