3. A treasure map is represented as a rectangular grid. Each grid location contains either a single treasure or nothing. The grid is represented using a matrix of Boolean values. If a cell in the grid contains a treasure then the value `true` is stored in the corresponding matrix location; otherwise, the value `false` is stored.

Consider the following declaration for the `TreasureMap` class.

```
class TreasureMap
{
  public:

    // ... constructors not shown

    bool HasTreasure(int row, int col) const;
    // postcondition: returns true if the cell at location (row, col)
    //                contains a treasure;
    //                returns false if location (row, col) is not within
    //                the bounds of the grid or if there is no treasure
    //                at that location

    int NumAdjacent(int row, int col) const;
    // precondition:  0 <= row < NumRows(); 0 <= col < NumCols()
    // postcondition: returns a count of the number of treasures in the
    //                cells adjacent to the location (row, col),
    //                horizontally, vertically, and diagonally

    int NumRows() const;
    // postcondition: returns the number of rows in the treasure map

    int NumCols() const;
    // postcondition: returns the number of columns in the treasure map

  private:

    apmatrix<bool> myGrid;
       // myGrid[r][c] being true indicates a treasure at (r, c)
       // the matrix is sized by the constructor
};
```

For example, suppose that the 6-by-9 grid shown below is a treasure map where the symbol 💰 in a cell indicates a treasure. In this example, `myGrid[2][3]` is `true` and `myGrid[1][2]` is `false`.

	0	1	2	3	4	5	6	7	8
0		💰	💰		💰		💰		
1		💰					💰		
2		💰		💰	💰			💰	💰
3	💰		💰		💰	💰			
4		💰			💰			💰	
5	💰			💰		💰			

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available to AP professionals at apcentral.collegeboard.com and to
students and parents at www.collegeboard.com/apstudents.

GO ON TO THE NEXT PAGE.

(a) Write the `TreasureMap` member function `HasTreasure`, which is described as follows. `HasTreasure` returns `true` if there is a treasure at the location `(row, col)`. If `(row, col)` is not within the bounds of the grid or if there is no treasure at that location, `HasTreasure` returns `false`.

For example, if `TreasureMap theMap` represents the treasure map shown at the beginning of the question, the following table gives the results of several calls to `HasTreasure`.

Function call	Value returned
`theMap.HasTreasure(0, 2)`	`true`
`theMap.HasTreasure(0, -1)`	`false`
`theMap.HasTreasure(2, 3)`	`true`
`theMap.HasTreasure(2, 2)`	`false`
`theMap.HasTreasure(4, 9)`	`false`

Complete function `HasTreasure` below.

```
bool TreasureMap::HasTreasure(int row, int col) const
// postcondition: returns true if the cell at location (row, col)
//                contains a treasure;
//                returns false if location (row, col) is not within
//                the bounds of the grid or if there is no treasure
//                at that location
```

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available to AP professionals at apcentral.collegeboard.com and to
students and parents at www.collegeboard.com/apstudents.

GO ON TO THE NEXT PAGE.

(b) Write the `TreasureMap` member function `NumAdjacent`, which is described as follows. `NumAdjacent` returns the number of treasures that are adjacent to a given location specified by `row` and `col`. To be adjacent, a treasure must be in one of the (at most) eight cells that border the given location horizontally, vertically, or diagonally; a treasure in the given location does not count as being adjacent.

The treasure map below is repeated for your convenience.

	0	1	2	3	4	5	6	7	8
0		💰	💰		💰		💰		
1		💰					💰		
2		💰		💰	💰			💰	💰
3	💰		💰		💰	💰			
4		💰			💰			💰	
5	💰			💰		💰			

For example, if `TreasureMap theMap` represents the treasure map shown above, the following table gives the results of several calls to `NumAdjacent`.

Function call	Value returned
`theMap.NumAdjacent(3, 3)`	5
`theMap.NumAdjacent(2, 4)`	3
`theMap.NumAdjacent(4, 7)`	0

In writing `NumAdjacent`, you may call `HasTreasure` specified in part (a). Assume that `HasTreasure` works as specified, regardless of what you wrote in part (a).

Complete function `NumAdjacent` below.

```
int TreasureMap::NumAdjacent(int row, int col) const
// precondition:  0 <= row < NumRows(); 0 <= col < NumCols()
// postcondition: returns a count of the number of treasures in the
//                cells adjacent to the location (row, col),
//                horizontally, vertically, and diagonally
```

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available to AP professionals at apcentral.collegeboard.com and to
students and parents at www.collegeboard.com/apstudents.

GO ON TO THE NEXT PAGE.

(c) Write free function `ComputeCounts`, which is described as follows. `ComputeCounts` returns a matrix of integers where the value at `(row, col)` is `9` if there is a treasure at location `(row, col)` in `theMap`. Otherwise, the value at `(row, col)` is the number of treasures adjacent to location `(row, col)`.

For example, the following shows the matrix that is returned as a result of calling `ComputeCounts` with the `TreasureMap aMap`.

`aMap`

	0	1	2	3	4
0		💰		💰	💰
1	💰				
2		💰	💰		

Matrix returned by the call `ComputeCounts(aMap)`

	0	1	2	3	4
0	2	9	2	9	9
1	9	4	4	3	2
2	2	9	9	1	0

In writing `ComputeCounts`, you may call any `TreasureMap` member function. Assume that all member functions of `TreasureMap` work as specified, regardless of what you wrote in parts (a) and (b).

Complete function `ComputeCounts` below.

```
apmatrix<int> ComputeCounts(const TreasureMap & theMap)
```

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available to AP professionals at apcentral.collegeboard.com and to
students and parents at www.collegeboard.com/apstudents.

GO ON TO THE NEXT PAGE.

4. This question involves reasoning about the code from the Marine Biology Case Study. A copy of the code is provided in the Appendix.

   The marine biologists want to study a species of fish that eats algae. Any position in the environment grid can contain zero or more units of algae. If there is any algae at a fish's location, the fish eats one unit of the algae and does not move; otherwise, the fish does not eat. If this is the third consecutive step in which the fish has not eaten, then the fish dies and is removed from the environment. If the fish does not eat and does not die, it moves to a position among its empty neighbors that contains the most algae.

   We represent the algae by adding a matrix of integers to the private data of the `Environment` class. This matrix is the same size as `myWorld`, and each entry represents the number of units of algae at that location. We add three public member functions to the `Environment` class, as well as modifying the `Environment` constructor to initialize `myAlgae`.

```
//  Added to the public section of class Environment

    void RemoveFish(const Position & pos);
    // precondition:  there is a fish at pos
    // postcondition: there is no fish at pos

    int NumAlgaeAt(const Position & pos) const;
    // precondition:  pos is a valid position in the environment
    // postcondition: returns the number of units of algae at pos

    void RemoveAlgae(const Position & pos, int numUnits);
    // precondition:  algae at position pos exceeds numUnits
    // postcondition: algae at position pos has been reduced by numUnits



//  Added to the private section of class Environment

    apmatrix<int> myAlgae;  // number of units of algae at each position
```

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available to AP professionals at apcentral.collegeboard.com and to
students and parents at www.collegeboard.com/apstudents.

GO ON TO THE NEXT PAGE.

AP® COMPUTER SCIENCE A
2003 SCORING GUIDELINES

Question 3

Part A: `HasTreasure` **1 pt**

+1
- **+1/2** attempt (must examine a position in `myGrid` and some boundary)
- **+1/2** correct

Part B: `NumAdjacent` **4 pts**

+1/2 declare and initialize counter

+1 Scan 3x3 block centered at this location, except this location
- **+1/2** attempt (must have nested loops or at least five individual cases)
- **+1/2** correct for outer eight

+1 Check `HasTreasure` for each location
- **+1/2** attempt (use loop indices in loop case,
  at least two different offsets from this location for non-loop)
- **+1/2** correct

+1/2 exclude this location from count
(may skip in scan or may count and subtract out)

+1/2 increment counter

+1/2 return counter

Part C: `ComputeCounts` **4 pts**

+1/2 declare result matrix

+1 scan over full map
- **+1/2** attempt (must have nested loops and reference map boundaries,
  not hard-coded constants)
- **+1/2** correct

+2 fill result matrix
- **+1/2** check `HasTreasure`
- **+1/2** assign 9 in true case
- **+1/2** call `NumAdjacent` in false case
- **+1/2** assign `NumAdjacent` in false case

+1/2 return result matrix

Usage for part C: -1 for two or more instances of missing `theMap`
-1 for using `myGrid` as the result matrix.

Copyright © 2003 by College Entrance Examination Board. All rights reserved.
Available at apcentral.collegeboard.com.