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

<table>
<thead>
<tr>
<th>Function call</th>
<th>Value returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>theMap.HasTreasure(0, 2)</td>
<td>true</td>
</tr>
<tr>
<td>theMap.HasTreasure(0, -1)</td>
<td>false</td>
</tr>
<tr>
<td>theMap.HasTreasure(2, 3)</td>
<td>true</td>
</tr>
<tr>
<td>theMap.HasTreasure(2, 2)</td>
<td>false</td>
</tr>
<tr>
<td>theMap.HasTreasure(4, 9)</td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function HasTreasure below.

```cpp
class TreasureMap {
public:
    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
        if ((row < 0 || row >= NumRows()) || (col < 0 || col >= NumCols()))
            return false;
        return m_yGrid[row][col];
    }
};
```

Part (b) begins on page 14.
Complete function `NumAdjacent` below.

```cpp
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
{
    int r, c, numTreasure = 0;
    for (r = row - 1; r <= row + 1; r++)
    {
        for (c = col - 1; c <= col + 1; c++)
        {
            if (HasTreasure(r, c))
                numTreasure += 1;
        }
    }
    if (HasTreasure(row, col))
        numTreasure -= 1;
    return numTreasure;
}
```

Part (c) begins on page 16.
Complete function `ComputeCounts` below.

```cpp
apmatrix<int> ComputeCounts(const TreasureMap & theMap) {
    apmatrix<int> newMap (theMap.NumRows(), theMap.NumCols());
    int r, c;
    for (r = 0; r < newMap.numrows(); r++)
    {
        for (c = 0; c < newMap.numcols(); c++)
        {
            if (theMap.HasTreasure(r, c))
                newMap[r][c] = 9;
            else
                newMap[r][c] = theMap.NumAdjacent(r, c);
        }
    }
    return newMap;
}
```
(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.

<table>
<thead>
<tr>
<th>Function call</th>
<th>Value returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>theMap.HasTreasure(0, 2)</td>
<td>true</td>
</tr>
<tr>
<td>theMap.HasTreasure(0, -1)</td>
<td>false</td>
</tr>
<tr>
<td>theMap.HasTreasure(2, 3)</td>
<td>true</td>
</tr>
<tr>
<td>theMap.HasTreasure(2, 2)</td>
<td>false</td>
</tr>
<tr>
<td>theMap.HasTreasure(4, 9)</td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function HasTreasure below.

```cpp
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
{
  if ((row >= 0) && (row < myGrid.NUMRows()) && (col >= 0)
      && (col < myGrid.NUMCols()) && (myGrid[Row][Col] == true))
    return true;
  else
    return false;
}
```

Part (b) begins on page 14.
Complete function NumAdjacent below.

```cpp
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
{
    INT COUNTER = 0;
    if (HasTreasure(row + 1, col))
        COUNTER++;
    if (HasTreasure(row - 1, col))
        COUNTER++;
    if (HasTreasure(row, col + 1))
        COUNTER++;
    if (HasTreasure(row, col - 1))
        COUNTER++;
    if (HasTreasure(row + 1, col + 1))
        COUNTER++;
    if (HasTreasure(row - 1, col - 1))
        COUNTER++;
    if (HasTreasure(row + 1, col - 1))
        COUNTER++;
    if (HasTreasure(row - 1, col + 1))
        COUNTER++;
    return COUNTER;
}
```

Part (c) begins on page 16.
Complete function `ComputeCounts` below.

```cpp
template<
    typename T
>
T ComputeCounts(const TreasureMap & theMap)
{
    // REMINDS ME OF MINESWEEPER!
    T mines(theMap.NumRows(), theMap.NumCols());
    for (int row = 0; row < theMap.NumRows(); row++)
        for (int col = 0; col < theMap.NumCols(); col++)
        {
            if (theMap.HasTreasure(row, col))
                mines[row][col] = 9;
            else
                mines[row][col] = theMap.NumAdjacent(row, col);
        }
    return mines;
}
```

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.

<table>
<thead>
<tr>
<th>Function call</th>
<th>Value returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>theMap.HasTreasure(0, 2)</td>
<td>true</td>
</tr>
<tr>
<td>theMap.HasTreasure(0, -1)</td>
<td>false</td>
</tr>
<tr>
<td>theMap.HasTreasure(2, 3)</td>
<td>true</td>
</tr>
<tr>
<td>theMap.HasTreasure(2, 2)</td>
<td>false</td>
</tr>
<tr>
<td>theMap.HasTreasure(4, 9)</td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function HasTreasure below.

```cpp
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
{
    if (theMap.myGrid[row][col] == True)
        return True;
    else
        return False;
}
```

Part (b) begins on page 14.
Complete function `NumAdjacent` below.

```cpp
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
{
    int counter = 0;
    if (theMap.myGrid[row + 1][col] == True)
        counter ++;
    if (theMap.myGrid[row + 1][col + 1] == True)
        counter ++;
    if (theMap.myGrid[row + 1][col - 1] == True)
        counter ++;
    if (theMap.myGrid[row][col - 1] == True)
        counter ++;
    if (theMap.myGrid[row][col + 1] == True)
        counter ++;
    if (theMap.myGrid[row - 1][col] == True)
        counter ++;
    if (theMap.myGrid[row - 1][col + 1] == True)
        counter ++;
    if (theMap.myGrid[row - 1][col - 1] == True)
        counter ++;
    return counter;
}
```

Part (c) begins on page 16.
Complete function \texttt{ComputeCounts} below.

\begin{verbatim}

\texttt{apmatrix<int> ComputeCounts(const TreasureMap & theMap)}

\texttt{apmatrix<int> Solution (theMap.NumRows(), theMap.NumCols());}

\texttt{for (int a = 0; a < theMap.NumRows(); a++)}
\texttt{\quad for (int b = 0; b < theMap.NumCols(); b++)}
\texttt{\quad \quad if (theMap.HasTreasure(a, b) == True)}
\texttt{\quad \quad \quad Solution[a][b] = 9;}
\texttt{\quad else}
\texttt{\quad \quad \quad Solution[a][b] = theMap.NumAdjacent(a, b);}

\texttt{return Solution;}

\end{verbatim}