AP® Computer Science A
2003 Sample Student Responses

These materials were produced by Educational Testing Service® (ETS®), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association whose mission is to prepare, inspire, and connect students to college and opportunity. Founded in 1900, the association is composed of more than 4,300 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 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®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit www.collegeboard.com

For the College Board's online home for AP professionals, visit AP Central at apcentral.collegeboard.com.

A3 A

(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
{
   if((row<0 || row>=NumRows()) || (col<0 || col>=NumCols())
      return false;
   return myGrid[row][col];
}
```

Part (b) begins on page 14.

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
{
    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++;
      }
    }
    if (HasTreasure(row,col))
      numTreasure--;
    return numTreasure;
}
```

Part (c) begins on page 16.

Complete function ComputeCounts below.

```
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;
}
```

GO ON TO THE NEXT PAGE.

A3 B

(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
{ 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.

```
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.

```
apmatrix<int> ComputeCounts(const TreasureMap & theMap)
{
  apMATRIX<INT> mines(theMap.NumRows(), theMap.NumCols())
  // REMINDS ME OF MINESWEEPER!
  FOR (INT Row1 = 0; Row1 < mines.NUMRows(); Row1++){
      for (INT col1=0; col1 < mines.NumCols(); col1++){
          if (theMap.HasTreasure(Row1, col1){
              mines[Row1][col1] = 9;
          }
          else {
              mines[Row1][col1] = theMap.NumAdjacent(Row1, col1,
          }
      }
  }
  RETURN MINES;
}
```

GO ON TO THE NEXT PAGE.

A3 C

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

Part (b) begins on page 14.

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
{
    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.

GO ON TO THE NEXT PAGE.

Complete function ComputeCounts below.

```
apmatrix<int> ComputeCounts(const TreasureMap & theMap)
{  apmatrix <int>  Solution ( theMap. NumRows (), theMap. NumCols ());

   for ( int a = 0 ;  a < theMap. NumRows (); a ++)
      for ( int b=0; b < theMap. NumCols (); b++)
         {
            if ( theMap. HasTreasure (a,b)  = True)
                  solution [a] [b] = 9;

            else
               solution [a] [b] = theMap. NumAdjacent (a,b);

         }

   return  solution;
}
```