AP Computer Science A
1999 Sample Student Responses

The materials included in these files are intended for non-commercial 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 face-to-face teaching purposes but may not mass distribute the materials, 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. This permission does not apply to any third-party copyrights contained herein.
(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile`. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file pattern, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
...x
.x.x.
..x.
...x.
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```java
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
    : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
      myColsOfBlocks(colsOfBlocks)
// precondition: inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
// the number of rows and columns of blocks that make up
// the quilt; myBlock has been resized and
// initialized to the block pattern from the
// stream inFile.
{
    int row, col;
    inFile >> row >> col;
    myBlock.resize(row, col);
    char x;
    for (int i = 0; i < row; i++)
        for (int k = 0; k < col; k++)
            {
            inFile >> x;
            myBlock[i][k] = x;
            }
}
```

Part (b) begins on page 6.
Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```cpp
void Quilt::PlaceFlipped(int startRow, int startCol,
                          amatrix<char> & qmat)
    // precondition: startRow ≥ 0; startCol ≥ 0;
    // startRow + myBlock.numrows() ≤ qmat.numrows();
    // startCol + myBlock.numcols() ≤ qmat.numcols();
    // postcondition: a flipped version of myBlock has been copied into the
    // matrix qmat with its upper-left corner at the position
    // startRow, startCol
{
    int r, c;

    for (r = 0; r < myBlock.numrows(); r++)
        for (c = 0; c < myBlock.numcols(); c++)
            qmat[startRow + r][startCol + c] = myBlock[myBlock.numrows() - r - 1][c];
}
```

Part (c) begins on page 8.
(c) Write the member function QuiltToMat, as started below. QuiltToMat returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If \( Q \) represents the example quilt, then the call \( Q::\text{QuiltToMat}() \) would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing QuiltToMat, you may call functions PlaceBlock and PlaceFlipped specified in part (b). Assume that PlaceBlock and PlaceFlipped work as specified, regardless of what you wrote in part (b).

Complete the member function QuiltToMat below.

```cpp
apmatrix<char> Quilt::QuiltToMat()
{
    apmatrix<char> mat(myBlock.numrows() * myRowsOfBlocks,
                        myBlock.numcols() * myColsOfBlocks);

    for (int k = 0; k < myRowsOfBlocks; k++)
        for (int p = 0; p < myColsOfBlocks; p++)
            if ((k + p) % 2 == 0)
                PlaceBlock(k * myBlock.numrows(), p * myBlock.numcols(), mat);
            else
                PlaceFlipped(k * myBlock.numrows(), p * myBlock.numcols(), mat);

    return mat;
} 
```
(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter `inFile`. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file `pattern`, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
  x.x.x
  .x.x.
  ..x.
  ..x.
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
    : myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
      myColsOfBlocks(colsOfBlocks)
// precondition: inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
// the number of rows and columns of blocks that make up
// the quilt; myBlock has been resized and
// initialized to the block pattern from the
// stream inFile.
```

```cpp
  
  int rows, cols;
  char le1;
  inFile >> rows;
  inFile >> cols;
  myBlock.resize(rows, cols);
  for (int i = 0; i < rows; i++)
  {
    for (int j = 0; j < cols; j++)
    {
      inFile >> le1;
      myBlock(i, j) = le1;
    }
  }

part (b) begins on page 14.
```

---

GO ON TO THE NEXT PAGE
Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```cpp
void Quilt::PlaceFlipped(int startRow, int startCol,
                          amatrix<char> & qmat)

// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: a flipped version of myBlock has been copied into the
//                matrix qmat with its upper-left corner at the position
//                startRow, startCol
{
    int r, c;

    for (r = 0; r < myBlock.numrows(); r++)
    {
        for (c = 0; c < myBlock.numcols(); c++)
        {
            qmat[startRow+r][startCol+c] = myBlock[myBlock.numrows()-1-r][myBlock.numcols()-1-c];
        }
    }
}
```

Part (c) begins on page 16.
(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If `Q` represents the example quilt, then the call `Q. QuiltToMat()` would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```cpp
apmatrix<char> Quilt::QuiltToMat()
{
    bool switch = true;
    for (int i = 0; i < myRows of Blocks; i++)
    {
        for (int j = 0; j < myCols of Blocks; j++)
        {
            if (switch)
                PlaceBlock (myBlock, numRows C) & i, myBlock, numCols C) Xj, this);
            else
                PlaceFlipped (myBlock, numRows (Xj, myBlock, numCols C) Xj, this);
            switch = !switch;
        }
    }
}```
(a) Write the code for the constructor that initializes a quilt, as started below. The constructor reads the block pattern for the main block from a file represented by the parameter inFile. You may assume the file is open and that the file contains the number of rows followed by the number of columns for the block, followed by the characters representing the pattern. For example, the file pattern, which contains the pattern for the first block in the quilt shown above, would look like this:

```
4 5
x...x
.x.x.
..x..
..x..
```

The constructor also sets the number of rows and columns of blocks which make up the entire quilt in the initializer list.

Complete the constructor below. Assume that the constructor is called only with parameters that satisfy its precondition.

```
Quilt::Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks)
: myBlock(0, 0), myRowsOfBlocks(rowsOfBlocks),
   myColsOfBlocks(colsOfBlocks)
// precondition: inFile is open, rowsOfBlocks > 0, colsOfBlocks > 0
// postcondition: myRowsOfBlocks and myColsOfBlocks are initialized to
//                the number of rows and columns of blocks that make up
//                the quilt; myBlock has been resized and
//                initialized to the block pattern from the
//                stream inFile.

  for (int row = 0; row < rowsOfBlocks; row++)
    for (int col = 0; col < colsOfBlocks; col++)
      inFile >> row;
    inFile >> col;
    for (int row = 0; row < rowsOfBlocks; row++)
      for (int col = 0; col < colsOfBlocks; col++)
        inFile >> myBlock[row][col];
```

Part (b) begins on page 14.
Complete the member function PlaceFlipped below. Assume that PlaceFlipped is called only with parameters that satisfy its precondition.

```cpp
void Quilt::PlaceFlipped(int startRow, int startCol,
                          apmatrix<char> & qmat)
    // precondition: startRow ≥ 0; startCol ≥ 0;
    //                startRow + myBlock.numrows() ≤ qmat.numrows();
    //                startCol + myBlock.numcols() ≤ qmat.numcols();
    // postcondition: a flipped version of myBlock has been copied into the
    //                matrix qmat with its upper-left corner at the position
    //                startRow, startCol
    {
        int r, c;

        for (r = 0; r < myBlock.numrows(); r++)
            for (c = 0; c < myBlock.numcols(); c++)
                qmat[startRow + myBlock.numrows() - r][startCol + c] = myBlock[r][c];
    }
```

Part (c) begins on page 16.
(c) Write the member function `QuiltToMat`, as started below. `QuiltToMat` returns a matrix representing the whole quilt in such a way that the main block alternates with the flipped version of the main block, as shown in the original example. If \( Q \) represents the example quilt, then the call \( Q.\text{QuiltToMat}() \) would return a matrix of characters with the given block placed starting with the upper-left corner at position 0, 0; the flipped block placed with its upper-left corner at position 0, 5; the given block placed with its upper-left corner at position 0, 10; the flipped block placed with its upper-left corner at position 4, 0, and so on.

In writing `QuiltToMat`, you may call functions `PlaceBlock` and `PlaceFlipped` specified in part (b). Assume that `PlaceBlock` and `PlaceFlipped` work as specified, regardless of what you wrote in part (b).

Complete the member function `QuiltToMat` below.

```cpp
apmatrix<чar> Quilt::QuiltToMat()
    PlaceBlock (0,0,M);
    PlaceFlipped (0,5,M);
    PlaceBlock (0,10,M);
    PlaceFlipped (4,0,M);
    PlaceBlock (4,5,M);
    PlaceFlipped (4,10,M);
    PlaceBlock (8,0,M);
    PlaceFlipped (8,5,M);
    PlaceBlock (8,10,M);
```