4. A patchwork quilt can be made by sewing together many blocks, all of the same size. Each individual block is made up of a number of small squares cut from fabric. A block can be represented as a two-dimensional array of nonblank characters, each of which stands for one small square of fabric. The entire quilt can also be represented as a two-dimensional array of completed blocks. The example below shows an array that represents a quilt made of 9 blocks (in 3 rows and 3 columns). Each block contains 20 small squares (of 4 rows by 5 columns). The quilt uses 2 different fabric squares, represented by the characters `'x'` and `'.'`. We consider only quilts where the main block alternates with the same block flipped upside down (i.e., reflected about a horizontal line through the block's center), as in the example below.

```
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
-----+-----+-----
..x..|x...x|..x..
..x..|.x.x.|..x..
.x.x.|..x..|.x.x.
x...x|..x..|x...x
-----+-----+-----
x...x|..x..|x...x
.x.x.|..x..|.x.x.
..x..|.x.x.|..x..
..x..|x...x|..x..
```

Consider the problem of storing and displaying information about a quilt.

The class `Quilt`, whose declaration is shown below, is used to keep track of the blocks for an entire quilt. Since the pattern is based on one block, we only store that block and the number of rows and columns of blocks. For the example shown above, we would store the upper left 4×5 block, 3 for the number of rows of blocks in the quilt and 3 for the number of columns of blocks in the quilt.

```
class Quilt
{
  public:
    Quilt(istream & inFile, int rowsOfBlocks, int colsOfBlocks);
    // constructor, given number of blocks in each row and column

    apmatrix<char> QuiltToMat();
    // returns a matrix with the entire quilt stored in it

  private:
    apmatrix<char> myBlock; // stores pattern for one block
    int myRowsOfBlocks;     // number of rows of blocks in the quilt
    int myColsOfBlocks;     // number of columns of blocks in the quilt

    void PlaceBlock(int startRow, int startCol,
                    apmatrix<char> & qmat);
    void PlaceFlipped(int startRow, int startCol,
                      apmatrix<char> & qmat);
};
```

(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.
```

(b) Write the private member function `PlaceFlipped`, as started below. `PlaceFlipped` is intended to place a flipped (upside-down) version of the block into the matrix `qmat` with the flipped block's upper left corner located at the `startRow, startCol` position in `qmat`.

For example, if quilt `Q` contains the block shown in part (a) and if `M` is a matrix large enough to hold the characters in the whole quilt, then the call

```
Q.PlaceFlipped(4, 10, M)
```

would place the flipped version of `Q`'s quilt block into matrix `M` as the third block in the second row of quilt blocks. This is the block whose upper-left corner is at position `M[4][10]`. In the diagram below, the upper-left corner of the flipped block being placed into `M` is circled.

```
x...x | ..x.. | x...x
.x.x. | ..x.. | .x.x.
..x.. | .x.x. | ..x..
..x.. | x...x | ..x..
------+-------+------
..x.. | x...x | ⊙.x..
..x.. | .x.x. | ..x..
.x.x. | ..x.. | .x.x.
x...x | ..x.. | x...x
------+-------+------
x...x | ..x.. | x...x
.x.x. | ..x.. | .x.x.
..x.. | .x.x. | ..x..
..x.. | x...x | ..x..
```

You may adapt the code of the private member function `PlaceBlock`, given below, which places the block (not inverted) into the matrix `qmat` with the block's upper left corner located at the `startRow, startCol` position.

```
void Quilt::PlaceBlock(int startRow, int startCol,
                       apmatrix<char> & qmat)
// precondition:  startRow ≥ 0; startCol ≥ 0;
//                startRow + myBlock.numrows() ≤ qmat.numrows();
//                startCol + myBlock.numcols() ≤ qmat.numcols();
// postcondition: 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[r][c];
    }
  }
}
```

Complete the member function `PlaceFlipped` below. Assume that `PlaceFlipped` is called only with parameters that satisfy its precondition.

```
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++)
    {


    }
  }
}
```

(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.

```
apmatrix<char> Quilt::QuiltToMat()
```

ADDITIONAL WORKSPACE

END OF EXAMINATION

- MAKE SURE THAT YOU HAVE COMPLETED THE IDENTIFICATION INFORMATION AS REQUESTED ON THE BACK COVER OF THIS BOOKLET.
- CHECK TO SEE THAT YOUR AP NUMBER APPEARS IN THE BOX ON THE BACK COVER.
- MAKE SURE THAT YOU HAVE USED THE SAME SET OF AP NUMBER LABELS ON ALL AP EXAMINATIONS YOU HAVE TAKEN THIS YEAR.