(a) Write the Window member function `IsInBounds`, as started below. `IsInBounds` checks whether a single point is in the window.

For example, for any 5-by-4 Window \( W \), the following table shows the results of several calls to `IsInBounds`.

<table>
<thead>
<tr>
<th>Call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>W.IsInBounds(0, 0)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>W.IsInBounds(2, 1)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>W.IsInBounds(4, 3)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>W.IsInBounds(5, 3)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>W.IsInBounds(3, -1)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>W.IsInBounds(8, 8)</code></td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function `IsInBounds` below.

```cpp
bool Window::IsInBounds(int row, int col) const
// postcondition: returns true if the point (row, col) is in this window;
// otherwise, returns false
{
    return (row >= 0 && row < myNumRows) &&
            (col >= 0 && col < myNumCols);}
```
Complete function ColorSquare below.

```c
void Window::ColorSquare(int ULrow, int ULcol, int N, int val)
  // postcondition: all points in this window that are also in the
  // N-by-N square with upper left corner
  // (ULrow, ULcol) have been set to val;
  // points in the square that are not in this
  // window are ignored
{
  int r, c;

  for (r = ULrow; r < ULrow + N; r++)
    for (c = ULcol; c < ULcol + N; c++)
      if (IsInBounds(r, c))
        myMat[r][c] = val;
}
```
Complete function Enlarge below.

```c
void Enlarge(Window & W, const Rectangle & rect, int factor)
// precondition:  factor > 0
{
    int r, c;

    int newRow, newCol;

    for (r = 1; r <= rect.numRows; r++)
    {
        for (c = 1; c <= rect.numCols; c++)
        {
            newCol = ((rect.numCols - c) * factor) + rect_ULcol;
            newRow = ((rect.numRows - r) * factor) + rect_ULrow;

            W.setColorSquare(newRow, newCol, factor, W.ValAt(rect.numRows-r+rect_ULrow),
                                      rect.numCols-c+rect_ULcol);
        }
    }
```
(a) Write the Window member function IsInBounds, as started below. IsInBounds checks whether a single point is in the window.

For example, for any 5-by-4 Window W, the following table shows the results of several calls to IsInBounds.

<table>
<thead>
<tr>
<th>Call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>W.IsInBounds(0, 0)</td>
<td>true</td>
</tr>
<tr>
<td>W.IsInBounds(2, 1)</td>
<td>true</td>
</tr>
<tr>
<td>W.IsInBounds(4, 3)</td>
<td>true</td>
</tr>
<tr>
<td>W.IsInBounds(5, 3)</td>
<td>false</td>
</tr>
<tr>
<td>W.IsInBounds(3, -1)</td>
<td>false</td>
</tr>
<tr>
<td>W.IsInBounds(8, 8)</td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function IsInBounds below.

```cpp
bool Window::IsInBounds(int row, int col) const
// postcondition: returns true if the point (row, col) is in this window;
// otherwise, returns false

if ((row > myNumRows) || (col > myNumCols))
    return false;
if ((row < 0) || (col < 0))
    return false;
else return true;
```
Complete function `ColorSquare` below.

```c++
void Window::ColorSquare(int ULrow, int ULcol, int N, int val)
// postcondition: all points in this window that are also in the
// N-by-N square with upper left corner
// (ULrow, ULcol) have been set to val;
// points in the square that are not in this
// window are ignored
{
    if (W.IsInBounds(ULrow, ULcol))
        for (int k = ULrow; k < ULrow+N; k++)
            for (int j = ULcol; j < ULcol+N; j++)
                if (W.IsInBounds(k, j))
                    W.myMat[k][j] = val;
}```
Complete function Enlarge below.

```c
void Enlarge(Window & W, const Rectangle & rect, int factor)
// precondition: factor > 0

for (int n = rect.ulrow + 1; n < rect.numrows)
    for (int m = rect.ulcol + 1; m < rect.numcols)
        k = W.valAt((tem.0, temp.m));
        W.colorSquare(n, m, factor, k);
```
(a) Write the Window member function `IsInBounds`, as started below. `IsInBounds` checks whether a single point is in the window.

For example, for any 5-by-4 Window W, the following table shows the results of several calls to `IsInBounds`.

<table>
<thead>
<tr>
<th>Call</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>W.IsInBounds(0, 0)</td>
<td>true</td>
</tr>
<tr>
<td>W.IsInBounds(2, 1)</td>
<td>true</td>
</tr>
<tr>
<td>W.IsInBounds(4, 3)</td>
<td>true</td>
</tr>
<tr>
<td>W.IsInBounds(5, 3)</td>
<td>false</td>
</tr>
<tr>
<td>W.IsInBounds(3, -1)</td>
<td>false</td>
</tr>
<tr>
<td>W.IsInBounds(8, 8)</td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function `IsInBounds` below.

```cpp
bool Window::IsInBounds(int row, int col) const
// postcondition: returns true if the point (row, col) is in this window;
// otherwise, returns false
{
  bool check = true;
  if (row < 0 || row >= myNumRows)
    check = false;
  else if (col < 0 || col >= myNumCols)
    check = false;
  return check;
}
```
Complete function ColorSquare below.

void Window::ColorSquare(int ULrow, int ULcol, int N, int val)
// postcondition: all points in this window that are also in the
// N-by-N square with upper left corner
// (ULrow, ULcol) have been set to val;
// points in the square that are not in this
// window are ignored

if(ISInBounds(ULrow, ULcol))
    for (int r = ULrow; r < ULrow + N; r++)
        for (int c = ULcol; c < ULcol + N; c++)
            myMat[r][c] = val;
void Enlarge(Window & W, const Rectangle & rect, int factor) {
    // precondition: factor > 0
    
    // Create a new vector list (rect.urow + rect.ulcol) x 0
    vector<int> list = list(rect.urow + rect.ulcol, 0);

    for (int i = rect.urow; i < rect.numrows(); i++) {
        for (int j = rect.ulcol(); j < rect.numcols(); j++) {
            for (int c = 0; c < list.length(); c++) {
                W. Color Square(rect.urow, rect.ulcol, list[i]);
            }
        }
    }
}