//////////////////////////////////////////////////////  
Instruction for operating the command line utility:  
//////////////////////////////////////////////////////

1. OpenCV needs to be installed
   * a version >= 3.1.
   * The bin directory (something like - C:\opencv\build\x64\vc14\bin) - needs to be added to Environment Variables.
2. Run the utility command line (interview\_excercise\_hole\_filling.exe) with parameters as follows:

* image path (example: submition\_files\input\_files\_for\_example\image1.jpg )
* mask path (example : submition\_files\input\_files\_for\_example\mask1.jpg )
* Z
* Epsilon
* conectivity type (expected values: 4 or 8)

Example of running:

> cd "the path to submition\_files"\submition\_files

> interview\_excercise\_hole\_filling.exe input\_files\_for\_example\\image1.jpg input\_files\_for\_example\\mask1.jpg 3 0.001 8

The output file is saved under the folder from which the command line was operated.

//////////////////////////////////////////////////////  
Directory path for code review  
//////////////////////////////////////////////////////

The project shared in an github. link:

* all "library" related files (.h and .cpp) are grouped under directory : hole\_filling\_'library'
* all command line utility related files (.h and .cpp) are grouped under directory : command\_line\_utility

//////////////////////////////////////////////////////  
Questions  
//////////////////////////////////////////////////////

1. If there are m boundary pixels and n pixels inside the hole. the complexity of the algorithm is: **O()**, for each hole pixel we sum over all pixels in the boundary**.**

Express m by n:

The worst case is when every pixel in the hole is connecting only with another 2 hole pixels in a way that they share as little neighbors as possible:  
For 4-conectivity its simply a line. and for 8-conectivity its a diagonal line.  
In this case **m = O()**. every hole pixel adds k neighbors to boundary. so we have n\*k boundary pixels

The best case is a hole shape that has the smallest perimeter per area -> a circle. If the hole pixels arranged in a circle like shape. with area of O().  
 The perimeter ( boundary ) is **m =** **Omega()**.

So total run time is:  
**Omega()  
O()**

1. Approximate algorithm in O(

Every hole pixel is estimated only by its neighbors. Instead of all boundary pixels. We visit from the most outer pixels ( closest to boundary ) up to the most internal, so that the "skeleton" pixels are the last to be visited.

Algorithm:

1. We go over the boundary pixels. And find the hole pixels that connect (neighbor) to them. Those hole pixels filled only by their neighbors (the boundary).
2. Now the hole was shrinking. And we update the new boundary - ( all the pixels we just filled).
3. We will repeat this process (1 + 2) until the entire hole is filled.

Run time is O( **.** explanation:

For each pixel, we visit all of its neighbors to see whether to put them in the new boundary for the next iteration, and in addition, to estimating the value of the pixel, we average over its neighbors => Every pixel require x (num neighbors) calculations.

)This algorithm is also implemented in the code(