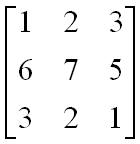
**Problem Statement :**

Implement a median filter from scratch using C++ .  Input should be an Image and the output should be a Linear filtered Image, Neat Documentation is expected with Code, Explanation, Input, and Output Image.

**Solution:**

Median filter also reduces the noise in an image like [low pass filter](https://www.programming-techniques.com/2013/01/low-pass-filters-blurring-in-image.html), but it is better than [low pass filter](https://www.programming-techniques.com/2013/01/low-pass-filters-blurring-in-image.html) in the sense that it preserves the edges and other details. The process of calculating the intensity of a central pixel is same as that of low pass filtering except instead of averaging all the neighbours, we sort the window and replace the central pixel with a median from the sorted window. For example, lets we have a window like this

[](https://1.bp.blogspot.com/-oG94gyRoPKU/URy0wxKURII/AAAAAAAAAi0/3d-mvzXZDDw/s1600/3x3window.jpg)

Now we sort the given window and get the sorted array as [1 1 2 2 3 3 5 6 7]. The median of this array is 4th element i.e 3. Now we replace the central element 7 with 4. That’s it.

**Code:**

#include<iostream>

#include<opencv2/imgproc/imgproc.hpp>

#include<opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

void insertionSort(int window[])

{

int temp, i , j;

for(i = 0; i < 9; i++)

{

temp = window[i];

for(j = i-1; j >= 0 && temp < window[j]; j--)

{

window[j+1] = window[j];

}

window[j+1] = temp;

}

}

int main()

{

Mat src, dst;

src = imread("book.png", CV\_LOAD\_IMAGE\_GRAYSCALE);

if( !src.data )

{ return -1; }

int window[9];

dst = src.clone();

for(int y = 0; y < src.rows; y++)

for(int x = 0; x < src.cols; x++)

dst.at<uchar>(y,x) = 0.0;

for(int y = 1; y < src.rows - 1; y++)

{

for(int x = 1; x < src.cols - 1; x++)

{

window[0] = src.at<uchar>(y - 1 ,x - 1);

window[1] = src.at<uchar>(y, x - 1);

window[2] = src.at<uchar>(y + 1, x - 1);

window[3] = src.at<uchar>(y - 1, x);

window[4] = src.at<uchar>(y, x);

window[5] = src.at<uchar>(y + 1, x);

window[6] = src.at<uchar>(y - 1, x + 1);

window[7] = src.at<uchar>(y, x + 1);

window[8] = src.at<uchar>(y + 1, x + 1);

insertionSort(window);

dst.at<uchar>(y,x) = window[4];

}

}

namedWindow("final");

imshow("final", dst);

namedWindow("initial");

imshow("initial", src);

waitKey();

return 0;

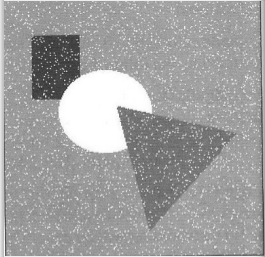
}

**Explanation:**

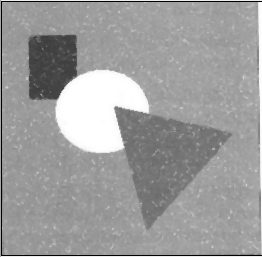
* Sort the window using insertion sort
* Insertion sort is best for this sorting
* Load an image
* Create a sliding window of size 9
* Pick up window element
* Sort the window to find median
* Assign the median to center element of the matrix

**Output:**

**Input image:**

****

**Output image:**

****