```cpp
#include "MorphologicalFilter.h"

namespace Vision
{
    MorphologicalFilter::MorphologicalFilter()  {   }

    MorphologicalFilter::MorphologicalFilter(const Mat &src)
    {
        OriginalImg = src;
        ProcessedImg.create(OriginalImg.size(), CV_8UC1);
    }

    MorphologicalFilter::~MorphologicalFilter() {}

    void MorphologicalFilter::Erosion(const Mat &src, Mat &dst, const Mat &mask)
    {
        OriginalImg = src;
        ProcessedImg.create(src.size(), CV_8UC1);
        Erosion(mask);
        dst = ProcessedImg;
    }

    void MorphologicalFilter::Erosion(const Mat &mask, bool chain)
    {
        // Exception handling
        CV_Assert(OriginalImg.depth() != sizeof(uchar));
        EMPTY_CHECK(OriginalImg);
        if (mask.cols % 2 == 0 || mask.cols < 3) { throw Exception::WrongKernelSizeException("Wrong Kernelsize columns!"); }
        if (mask.rows % 2 == 0 || mask.rows < 3) { throw Exception::WrongKernelSizeException("Wrong Kernelsize rows!"); }

        // make Pointers
        uchar *O;
        CHAIN_PROCESS(chain, O, uchar);
        uchar *P = ProcessedImg.data;

        // Init the relevant data
        uint32_t nCols = OriginalImg.cols;
        uint32_t nRows = OriginalImg.rows;
        uint32_t nData = nCols * nRows;
        uint32_t hKsizeCol = (mask.cols / 2);
        uint32_t hKszieRow = (mask.rows / 2);
```

```
42          uint32_t pEnd = nData - hKsizeCol - (hKszieRow * nCols);
43          uint32_t i = hKsizeCol + (hKszieRow * nCols);
44          uint32_t j = 0;
45          int currentKRow = -hKszieRow;
46          int currentKcol = -hKsizeCol;
47          bool isEroded = false;
48          uint32_t nKData = mask.cols * mask.rows;
49          uchar *nRow = GetNRow(nData, hKsizeCol, nCols, nRows);
50          uchar *nKRow = GetNRow(nKData, 0, mask.cols, mask.rows);
51
52          while (i < pEnd)
53          {
54              // Checks if pixel isn't a border pixel and progresses to the new row
55              if (nRow[i] == 1) { i += mask.cols; }
56
57              /*Loops through the mask to check if pixel is kept or deleted, check the zero's first because they're a multitude compared
                    with the ones and use
58              a else if loop so the other conditions are only checked  when it's not a zero*/
59              if (O[i] == 0) { P[i] = 0; }
60              else if (O[i])
61              {
62                  //Checks for each mask pixel that is set to one if the corresponding pixel is one
63                  j = 0;
64                  currentKRow = -hKszieRow;
65                  currentKcol = -hKsizeCol - 1;
66                  isEroded = false;
67                  while (j < nKData && !isEroded)
68                  {
69                      // Checks if pixel isn't a border pixel from the kernel
70                      currentKcol += 1;
71                      if (nKRow[j] == 1)
72                      {
73                          currentKRow++;
74                          currentKcol = -hKsizeCol;
75                      }
76
77                      // If one of the pixels is different then the corresponding mask pixel set the processed pixel to zero and exit
                            loop
78                      if (mask.data[j] != O[i + (currentKRow * nCols) + currentKcol])
79                      {
80
```

```
81                                  isEroded = true;
82                              }
83                              j++;
84                          }
85                          // If all pixel in the ROI image correspond with the mask pixels set the processed pixel to one
86                          if (!isEroded) { P[i] = 1; }
87                      }
88                      else { throw Exception::PixelValueOutOfBoundException(); } // Unexpected value throw exception
89                      i++;
90                  }
91              }
92          }
```