以下所有的 kernel 皆為 octagonal 3-5-5-5-3 kernel，以二維 array 表示之。

```
int kernel[21][2] = { {-2, -1}, {-2, 0}, {-2, 1},
                      {-1, -2}, {-1, -1}, {-1, 0}, {-1, 1}, {-1, 2},
                      {0, -2}, {0, -1}, {0, 0}, {0, 1}, {0, 2},
                      {1, -2}, {1, -1}, {1, 0}, {1, 1}, {1, 2},
                      {2, -1}, {2, 0}, {2, 1} };
```

1. Dilation

在Main Function中：

```
//進行Binary Dilation
Mat b_dilation = BinaryDilation(b_img, kernel, widthLimit, heightLimit);
```

```
template<typename T, size_t M, size_t N>
Mat BinaryDilation(Mat img, T (&kernel)[M][N], int widthLimit, int heightLimit) {
    //創建新圖準備處理
    Mat result(heightLimit, widthLimit, CV_8U, Scalar(0));

    for (int height = 0; height < heightLimit; height++) {
        for (int width = 0; width < widthLimit; width++) {
```

```
                if (img.at<uchar>(height,width) > 0) {
                    for (int i = 0; i < M; i += 1) {
                        if (kernel[i][0] + height >= 0 && kernel[i][0] + height < heightLimit
                            && kernel[i][1] + width >= 0 && kernel[i][1] + width < widthLimit)
                            result.at<uchar>(kernel[i][0] + height, kernel[i][1] + width) = 255;
                    }
                }
            }
        }

        return result;
}
```

解釋：開一新圖 result，利用 nested loop 跑完整張圖。因題目要求在 white pixels 上處理，故對於 itensity 為 255 的 pixel 進行處理。
當目前的 pixel 為 255 時在 kernel（這邊傳入的是 octagonal 3-5-5-5-3 kernel）套上後不超出邊界的情況下，將所有被罩住的 pixels 指派為 255（在 result 中相對應的 pixel）。
當所有 pixels 處理完畢，回傳 result。

2. Erosion

在Main Function中：

```
    //進行Binary Erosion
    Mat b_erosion = BinaryErosion(b_img, kernel, widthLimit, heightLimit);
```

```
Mat BinaryErosion(Mat img, T(&kernel)[M][N], int widthLimit, int heightLimit) {
    //創建新圖準備處理
    Mat result(heightLimit, widthLimit, CV_8U, Scalar(0));

    for (int height = 0; height < heightLimit; height++) {
        for (int width = 0; width < widthLimit; width++) {

            bool exist;

            if (img.at<uchar>(height, width) > 0) {
                exist = true;

                for (int i = 0; i < M; i += 1) {
                    if (kernel[i][0] + height < 0 || kernel[i][0] + height >= heightLimit
                        || kernel[i][1] + width < 0 || kernel[i][1] + width >= widthLimit
                        || img.at<uchar>(kernel[i][0] + height, kernel[i][1] + width) == 0) {
                        exist = false;
                        break;
                    }
                }

                if (exist)
                    result.at<uchar>(height, width) = 255;
            }
        }
    }

    return result;
}
```

解釋：開一新圖 result，利用 nested loop 跑完整張圖。因題目要求在 white pixels 上處理，故對於 itensity 為 255 的 pixel 進行處理。

設一個 boolean 變數為 true，當目前的 pixel 的 itensity 為 255 時在 kernel（這邊傳入的是 octagonal 3-5-5-5-3 kernel）套上後，若有任一 pixel 超出了 kernel，也就是原圖中原點位置無法容納下此 kernel，則將此 boolean 改為 false 並跳出 loop。

在內圈 loop 結束後，針對目前的 pixel 是否保留的結果儲存至 result 中。

當所有的 pixels 皆處理完畢，回傳 result。

3. Opening

在Main Function中：

```
//進行Binary Opening
Mat b_opening = BinaryOpening(b_img, kernel, widthLimit, heightLimit);
```

```
template<typename T, size_t M, size_t N>
Mat BinaryOpening(Mat img, T(&kernel)[M][N], int widthLimit, int heightLimit) {
    return BinaryDilation(BinaryErosion(img, kernel, widthLimit, heightLimit), kernel, widthLimit,
heightLimit);
}
```

Opening 相當於對原圖進行 Erosion 後再進行 Dilation。

4. Closing

在 Main Function 中：

```
//進行Binary Opening
Mat b_closing = BinaryClosing(b_img, kernel, widthLimit, heightLimit);
```

```
template<typename T, size_t M, size_t N>
Mat BinaryClosing(Mat img, T(&kernel)[M][N], int widthLimit, int heightLimit) {
    return BinaryErosion(BinaryDilation(img, kernel, widthLimit, heightLimit), kernel, widthLimit,
heightLimit);
}
```

Opening 相當於對原圖進行 Dilation 後再進行 Erosion。

5. Hit-and-miss transform

以下所用的 J_kernel 和 K_kerne 皆與課本偵測 upper-right corner 的 L 形 kernel 相同，並以二維 array 表示之。

```
int J_kernel[3][2] = { {0, -1}, {0, 0}, {1, 0} };
int K_kernel[3][2] = { {-1, 0}, {-1, 1}, {0, 1} };
```

在 Main Function 中：

```
//進行Hit and Miss transform
Mat b_HaM = BinaryHitAndMiss(b_img, J_kernel, K_kernel, widthLimit, heightLimit);
```

```
Mat BinaryHitAndMiss(Mat img, T(&J_kernel)[M][N], T(&K_kernel)[M][N], int widthLimit, int heightLimit) {
    //創建新圖準備處理
    Mat result(heightLimit, widthLimit, CV_8U, Scalar(255));

    //算出補圖
    Mat complement(heightLimit, widthLimit, CV_8U, Scalar(0));
    for (int height = 0; height < heightLimit; height++)
        for (int width = 0; width < widthLimit; width++)
            complement.at<uchar>(height, width) = 255 - img.at<uchar>(height, width);

    //進行Binary Dilation
    Mat img_erosion = BinaryErosion(img, J_kernel, widthLimit, heightLimit);
```

```
    //進行Binary Erosion
    Mat com_erosion = BinaryErosion(complement, K_kernel, widthLimit, heightLimit);

    for (int height = 0; height < heightLimit; height++)
        for (int width = 0; width < widthLimit; width++) {
            if (img_erosion.at<uchar>(height, width) == 0 && com_erosion.at<uchar>(height, width) == 0)
                result.at<uchar>(height, width) = 0;
        }

    return result;
}
```

解釋：先對原圖 img 算出補圖，即 255 減去 img 中對應 pixel 的 itensity，創造出補圖 complement。

再來令 img_ erosion 為原圖 img 以 J_kernel 進行 erosion 的結果、令 com_ erosion 為補圖 complement 以 K_kernel 進行 erosion 的結果。

其中 J 和 K 皆與課本偵測 upper-right corner 的 L 形 kernel 相同。

接著利用 nested loop，看看兩者之間有交集的地方，將結果處存在 result 中，其中若 img_ erosion 和 complement 中的 itensity 同時為 0（有值）者，即代表該處 pixel 有值。

所有 pixels 皆處理完畢後，回傳 result。