```
// image.h

#pragma once

#include <string.h>

// image.h
typedef struct image {
	int w;
	int h;
	int c;
	unsigned char* data;
} image;

image load_image(const char* filename);
image make_image(int w, int h, int c);
image make_empty_image(int w, int h, int c);
image RGBtoIntensity(image im);

image Intensity2RGB(image im);




// *******************************************************************************
// image.cpp

#include "image.h"

#define STB_IMAGE_IMPLEMENTATION
#include "stb/include/stb_image.h"
```

```
image load_image(const char* filename)
{
    int w, h, c; // width , height, channel
    int channel = 3;
    //w = width, h = height, c = # 8 - bit components per pixel ...
    unsigned char* data = stbi_load(filename, &w, &h, &c, channel);    // without OpenCV

    if (!data) {
        exit(EXIT_FAILURE);
    }

    image out;
    out.data = data;
    out.h = h;
    out.w = w;
    out.c = c;
    return out;
}//load_image

void Free(image im)
{
    delete[] im.data;
}

image RGBtoIntensity(image im)
{
    image raw;
    raw.data = new unsigned char[im.h * im.w]; // height*weight kadar yer aç
    raw.w = im.w;
    raw.h = im.h;
    raw.c = 1; // intensity-gray level'a çek, tek boyut
    long bufpos = 0;
    long newpos = 0;
    for (int row = 0; row < im.h; row++)
    {
        for (int column = 0; column < im.w; column++)
        {
            newpos = row * im.w + column;
            bufpos = row * im.w * im.c + column * im.c;
            raw.data[newpos] = unsigned char(0.30 * im.data[bufpos] + 0.59 * im.data[bufpos + 1] + 0.11 * im.data[bufpos + 2]);
        }
    }
    return raw;
}


image Intensity2RGB(image im) {
    image rgb;
    rgb.data = new unsigned char[im.h * im.w * 3]; // R, G, B için 3 kanal
    rgb.w = im.w;
    rgb.h = im.h;
    rgb.c = 3; // RGB formatında çıktı

    long bufpos = 0;
    long newpos = 0;
    for (int row = 0; row < im.h; row++) {
        for (int column = 0; column < im.w; column++) {
            newpos = row * im.w + column;
            bufpos = newpos * 3;
            unsigned char intensity = im.data[newpos];

            rgb.data[bufpos] = intensity / 0.3;        // R kanalına intensity değerini kopyala
            rgb.data[bufpos + 1] = intensity / 0.59;     // G kanalına intensity değerini kopyala
            rgb.data[bufpos + 2] = intensity / 0.11;     // B kanalına intensity değerini kopyala
        }
    }

    return rgb;
}
```

```
// clustering.h

#include "image.h"

int* Histogram(image im);

float* KMeans_Euclidean(image im, int k);

image KBasedSegmentation(image im, float* kmean, int k);

unsigned char EuclideanDistance(float data, float* kmeans, int k);


//******************************************************************************
// clustering.cpp

#include "clustering.h"
#include <iostream>
#include <cmath>

int* Histogram(image im)
{
	int* hist;
	if (im.c == 1)
	{
		hist = new int[256];

		// içeriği temizle
		for (int i = 0; i < 256; i++)
			hist[i] = 0;
```

```
                for (int i = 0; i < im.h * im.w; i++)
                {
                hist[im.data[i]]++;
                }

                return hist;
        }
        else hist = NULL;
}



unsigned char EuclideanDistance(float data, float* kmeans, int k)
{
  if (k != 2)
  {
    return 0;
  }

  float distance0 = std::abs(data - kmeans[0]); // İlk küme merkezine olan uzaklık
  float distance1 = std::abs(data - kmeans[1]); // İkinci küme merkezine olan uzaklık

  if (distance0 <= distance1)
  {
    return 1; // İlk küme merkezine daha yakınsa 0 döndür
  }
  else
  {
    return 2; // İkinci küme merkezine daha yakınsa 1 döndür
  }
}
```

```
float* KMeans_Euclidean(image im, int k)
{
    int* hist = Histogram(im);
    int MaxIntensity = 256;
    if (k == 2)
    {
        float Tlow = 20, Thigh = 150;
        float Tlow_new, Thigh_new;
        // Label each intensity
        float low, high, number_low, number_high;
        bool State = true;
        while (State)
        {
            low = high = number_low = number_high = 0.0;
            for (int i = 0; i < MaxIntensity; i++)
            {
     if (std::abs(i - Tlow) <= std::abs(i - Thigh))
     {
      low += (hist[i] * i);
      number_low += hist[i];
     }
     else
     {
      high += (hist[i] * i);
      number_high += hist[i];
     }
   }

   // Yeni eşik değerlerini hesapla
   Tlow_new = low / number_low;
```

```
            Thigh_new = high / number_high;

            // Eşik değerleri değişiklik oranını kontrol et
            if (std::abs(Tlow - Tlow_new) < 1.0 && std::abs(Thigh - Thigh_new) < 1.0)
            {
                State = false; // Değişiklik oranı yeterince küçükse döngüden çık
            }
            else
            {
                Tlow = Tlow_new;
                Thigh = Thigh_new;
            }
        }

        // Küme merkezlerini oluştur
        float* clusterCenters = new float[k];
        clusterCenters[0] = Tlow;
        clusterCenters[1] = Thigh;



        delete[] hist;
        return clusterCenters;
    }

    return nullptr; // Kümeleri sadece 2 olarak destekliyoruz, başka bir k değeri girildiğinde nullptr döndür
}



image KBasedSegmentation(image im, float* kmean, int k)
```

```
{
    float* means = new float[2]; // kmeans değerleri, back-fore yapıldığı için 2 adet değer içerir
    means[0] = 0.0;
    means[1] = 0.0;

    means[0] = kmean[0];
    means[1] = kmean[1];
    std::cout << "tlow: " << means[0] << std::endl;
    std::cout << "thigh: " << means[1] << std::endl;

    unsigned char cluster = -1; // bu böyle kalmalı mı bilmiyorum?
    unsigned char binary_1 = 255; // background
    unsigned char binary_0 = 0; // foreground , object


    for (int i = 0; i < im.h * im.w; i++)
    {
        cluster = EuclideanDistance(im.data[i], means, 2);

        if (int(cluster) == 1)
        {
            im.data[i] = binary_1;
        }
        else
        {
            if (int(cluster) == 2)
            {
                im.data[i] = binary_0;
            }
        }
    }
    image binary_im;
    binary_im.w = im.w;
    binary_im.h = im.h;
    binary_im.c = im.c;
    binary_im.data = im.data;

    return binary_im;
}
```

```
// morphology.h

#pragma once

#include "image.h"
#include "iostream"

// opening;



image erosion(image im, int struct_matris_dim);

image dilation(image im, int struct_matris_dim);




image opening(image im, int struct_matris_dim);

image closing(image im, int struct_matris_dim);




image edge_detection(image im, int struct_matris_dim);

image & labeling(image &im);



image& regionFilling(image& im);
```

```
// ****************************************************************************

// morphology.cpp
#include "morphology.h"



image complement(image im)
{
    int image_size = im.h * im.w;

    unsigned char binary_0 = 0;
    unsigned char binary_1 = 255;

    unsigned char* complement_data = new unsigned char[image_size];
    for (int i = 0; i < image_size; i++)
    {
        complement_data[i] = (im.data[i] == binary_0) ? binary_1 : binary_0; // Complementi hesapla
        im.data[i] = complement_data[i];
    }

    return im;

}



image erosion(image im, int struct_matris_dim)
{
    int im_column = im.w;
    int im_row = im.h;
    unsigned char* im_data = im.data;

    int m_dim = struct_matris_dim; // 3 ile denendi  // matrisin kaça kaçlık bir kare dimension olacağının atanması
    int m_size = m_dim * m_dim; // matris boyutu
    int image_size = im_column * im_row; //imajın boyutu

    unsigned char* new_data = new unsigned char[image_size]; // erosion uygulanan binary resmin yeni değerlerinin girileceği dizi
    for (int i = 0; i < image_size; i++)
    {
        new_data[i] = 0;
    }
    unsigned char* current_region = new unsigned char[m_size]; // resmin işlenecek pikseli ve o pikselin komşularını tutacak matris/dizi
    for (int i = 0; i < m_size; i++)
    {
        current_region[i] = 0;
    }


    unsigned char* erosion_matris = new unsigned char[m_size]; // erosion uygulanacak yapısal eleman matrisi/dizisi
    for (int i = 0; i < m_size; i++)
    {
        erosion_matris[i] = 255;
    }


    int* ands_result = new int[m_size]; // yapısal eleman ve resmin hedef bölgesinin or işlemi sonuçlarını tutacak matris /dizi
```

```
        for (int i = 0; i < m_size; i++)
        {
                ands_result[i] = 1;
        }
        int and_result = 1; // or işlemi sonucunun and işlemi yapıldıktan sonraki sonucunu tutacak
değişken




        // Komşu piksellerin indislerini hesapla
        for (int row = 0; row < im_row; row++) {
                for (int col = 0; col < im_column; col++) {

                        // sadece beyaz olanlar üzerinde işlem yap
                        if (int(im_data[row * im_column + col]) == 255)
                        {
                                and_result = 1; // Her piksel için and_result'i sıfırla

                                // Kenar piksel kontrolü
                                bool isLeftEdge = (col == 0);
                                bool isRightEdge = (col == im_column - 1);
                                bool isTopEdge = (row == 0);
                                bool isBottomEdge = (row == im_row - 1);

                                for (int r = 0; r < m_dim; r++) {
                                        for (int c = 0; c < m_dim; c++) {
                                                int imgRow = row - 1 + r;
                                                int imgCol = col - 1 + c;

                                                // İndislerin sınırlarını kontrol et
                                                if (imgRow >= 0 && imgRow < im_row && imgCol >=
0 && imgCol < im_column) {
```

```
                            current_region[r * m_dim + c] =
im_data[imgRow * im_column + imgCol];
                        }
                        else {
                            // Kenar piksel kontrolü
                            if (isLeftEdge && c == 0) { // buralara 0 ata bir
de
                                current_region[r * m_dim + c] = 0; //
Sol kenar pikseli için 1 değeri atanır
                            }
                            else if (isRightEdge && c == m_dim - 1) {
                                current_region[r * m_dim + c] = 0; //
Sağ kenar pikseli için 1 değeri atanır
                            }
                            else if (isTopEdge && r == 0) {
                                current_region[r * m_dim + c] = 0; // !
// Üst kenar pikseli için 1 değeri atanır
                            }
                            else if (isBottomEdge && r == m_dim - 1) {
                                current_region[r * m_dim + c] = 0; //
Alt kenar pikseli için 1 değeri atanır
                            }
                            else {
                                // İndis geçerli değil, dışarıda kalan
bölgeler için isteğe bağlı işlemler yapılabilir
                                // Örneğin, -1 veya farklı bir değer
atanabilir
                                current_region[r * m_dim + c] = 0;
                            }
                        }
                    }
                }

                for (int i = 0; i < m_size; i++)
                { // yapısal elaman or image hedef bölgesi
                    ands_result[i] = int(current_region[i]) & int(erosion_matris[i]);
                }
                for (int i = 0; i < m_size; i++)
                { // erosion sonucu değer
                    and_result = and_result & ands_result[i];
                }

                // eğer erosion sonucu 0 ise hedef pikselin değeri azaltılır
                if (and_result == 0)
                {
                    new_data[row * im_column + col] = 0; // !

                }
                else
                {
                    new_data[row * im_column + col] = 255;
                }
                // değilse aynı kalır
            }
        }
    }

    image erosion_image;
    erosion_image.h = im_row;
    erosion_image.w = im_column;
    erosion_image.c = im.c;
    erosion_image.data = new unsigned char[image_size];

    for (int i = 0; i < image_size; i++)
    {
```

```
			erosion_image.data[i] = new_data[i];
		}

		delete[] im_data;
		delete[] current_region;
		delete[] erosion_matris;
		delete[] ands_result;
		delete[] new_data;

		return erosion_image;
}

image dilation(image im, int struct_matris_dim)
{
		int im_column = im.w;
		int im_row = im.h;
		unsigned char* im_data = im.data;

		int m_dim = struct_matris_dim; // 3 ile denendi  // matrisin kaça kaçlık bir kare dimension
olacağının atanması
		int m_size = m_dim * m_dim; // matris boyutu
		int image_size = im_column * im_row; //imajın boyutu

		unsigned char* new_data = new unsigned char[image_size]; // dilation uygulanan binary
resmin yeni değerlerinin girileceği dizi
		for (int i = 0; i < image_size; i++)
		{
			new_data[i] = 0;
		}
		unsigned char* current_region = new unsigned char[m_size]; // resmin işlenecek pikseli ve o
pikselin komşularını tutacak matris/dizi
		for (int i = 0; i < m_size; i++)
```

```
    {
        current_region[i] = 0;
    }
    unsigned char* dilation_matris = new unsigned char[m_size]; // dilation uygulanacak yapısal eleman matrisi/dizisi
    for (int i = 0; i < m_size; i++)
    {
        dilation_matris[i] = 0;
    }

    int* ors_result = new int[m_size]; // yapısal eleman ve resmin hedef bölgesinin or işlemi sonuçlarını tutacak matris /dizi
    for (int i = 0; i < m_size; i++)
    {
        ors_result[i] = 0;
    }
    int or_result = 0; // or işlemi sonucunun and işlemi yapıldıktan sonraki sonucunu tutacak değişken

    // Komşu piksellerin indislerini hesapla
    for (int row = 0; row < im_row; row++) {
        for (int col = 0; col < im_column; col++) {

            or_result = 0; // Her piksel için and_result'i sıfırla

            // Kenar piksel kontrolü
            bool isLeftEdge = (col == 0);
            bool isRightEdge = (col == im_column - 1);
            bool isTopEdge = (row == 0);
            bool isBottomEdge = (row == im_row - 1);

            for (int r = 0; r < m_dim; r++) {
                for (int c = 0; c < m_dim; c++) {
                    int imgRow = row - 1 + r;
                    int imgCol = col - 1 + c;

                    // İndislerin sınırlarını kontrol et
                    if (imgRow >= 0 && imgRow < im_row && imgCol >= 0 && imgCol < im_column) {
                        current_region[r * m_dim + c] = im_data[imgRow * im_column + imgCol];
                    }
                    else {
                        // Kenar piksel kontrolü
                        if (isLeftEdge && c == 0) { // buralara 0 ata bir de
                            current_region[r * m_dim + c] = 0; // Sol kenar pikseli için 1 değeri atanır
                        }
                        else if (isRightEdge && c == m_dim - 1) {
                            current_region[r * m_dim + c] = 0; // Sağ kenar pikseli için 1 değeri atanır
                        }
                        else if (isTopEdge && r == 0) {
                            current_region[r * m_dim + c] = 0; // ! // Üst kenar pikseli için 1 değeri atanır
                        }
                        else if (isBottomEdge && r == m_dim - 1) {
                            current_region[r * m_dim + c] = 0; // Alt kenar pikseli için 1 değeri atanır
                        }
                        else {
                            // İndis geçerli değil, dışarıda kalan bölgeler için isteğe bağlı işlemler yapılabilir
                            // Örneğin, -1 veya farklı bir değer atanabilir
                            current_region[r * m_dim + c] = 0;
                        }
                }
            }
        }

        for (int i = 0; i < m_size; i++)
        { // yapısal elaman or image hedef bölgesi
            ors_result[i] = int(current_region[i]) | int(dilation_matris[i]);
        }
        for (int i = 0; i < m_size; i++)
        { // dilation sonucu değer
            or_result = or_result | ors_result[i];
        }

        // eğer dilation sonucu 0 ise hedef pikselin değeri azaltılır
        if (or_result == 0)
        {
            new_data[row * im_column + col] = 0; // !
        }
        else
        {
            new_data[row * im_column + col] = 255;
        }
        // değilse aynı kalır

    }
}

image dilation_image;
dilation_image.h = im_row;
dilation_image.w = im_column;
dilation_image.c = im.c;
```

```
    dilation_image.data = new unsigned char[image_size];

    for (int i = 0; i < image_size; i++)
    {
        dilation_image.data[i] = new_data[i];
    }

    delete[] im_data;
    delete[] current_region;
    delete[] dilation_matris;
    delete[] ors_result;
    delete[] new_data;

    return dilation_image;
}




image opening(image im, int struct_matris_dim)
{
    image im_erosion;
    im_erosion = erosion(im, struct_matris_dim);

    image im_dilation;
    im_dilation = dilation(im_erosion, struct_matris_dim);

    return im_dilation;
}

image closing(image im, int struct_matris_dim)
```

```
{
    image im_dilation;
    im_dilation = dilation(im, struct_matris_dim);


    image im_erosion;
    im_erosion = erosion(im_dilation, struct_matris_dim);



    return im_erosion;
}





image edge_detection(image im, int struct_matris_dim)
{
    int im_column = im.w;
    int im_row = im.h;
    unsigned char* im_data = im.data;


    int m_dim = struct_matris_dim; // 3 ile denendi  // matrisin kaça kaçlık bir kare dimension olacağının atanması
    int m_size = m_dim * m_dim; // matris boyutu
    int image_size = im_column * im_row; //imajın boyutu


    unsigned char* new_data = new unsigned char[image_size]; // dilation uygulanan binary resmin yeni değerlerinin girileceği dizi
    for (int i = 0; i < image_size; i++)
    {
        new_data[i] = 0;
    }
    unsigned char* current_region = new unsigned char[m_size]; // resmin işlenecek pikseli ve o pikselin komşularını tutacak matris/dizi
    for (int i = 0; i < m_size; i++)
    {
        current_region[i] = 0;
    }


    unsigned char* dilation_matris = new unsigned char[m_size]; // dilation uygulanacak yapısal eleman matrisi/dizisi
    for (int i = 0; i < m_size; i++)
    {
        dilation_matris[i] = 0;
    }


    int* ors_result = new int[m_size]; // yapısal eleman ve resmin hedef bölgesinin or işlemi sonuçlarını tutacak matris /dizi
    for (int i = 0; i < m_size; i++)
    {
        ors_result[i] = 0;
    }
    int or_result = 0; // or işlemi sonucunun and işlemi yapıldıktan sonraki sonucunu tutacak değişken



    // Komşu piksellerin indislerini hesapla
    for (int row = 0; row < im_row; row++) {
        for (int col = 0; col < im_column; col++) {

            if (int(im_data[row * im_column + col]) == 0)
            {
                or_result = 0; // Her piksel için and_result'i sıfırla

                // Kenar piksel kontrolü
                bool isLeftEdge = (col == 0);
                bool isRightEdge = (col == im_column - 1);
                bool isTopEdge = (row == 0);
                bool isBottomEdge = (row == im_row - 1);

                for (int r = 0; r < m_dim; r++) {
                    for (int c = 0; c < m_dim; c++) {
                        int imgRow = row - 1 + r;
                        int imgCol = col - 1 + c;

                        // İndislerin sınırlarını kontrol et
                        if (imgRow >= 0 && imgRow < im_row && imgCol >=
0 && imgCol < im_column) {
                            current_region[r * m_dim + c] =
im_data[imgRow * im_column + imgCol];
                        }
                        else {
                            // Kenar piksel kontrolü
                            if (isLeftEdge && c == 0) { // buralara 0 ata bir
de
                                current_region[r * m_dim + c] = 0; //
Sol kenar pikseli için 1 değeri atanır
                            }
                            else if (isRightEdge && c == m_dim - 1) {
                                current_region[r * m_dim + c] = 0; //
Sağ kenar pikseli için 1 değeri atanır
                            }
                            else if (isTopEdge && r == 0) {
                                current_region[r * m_dim + c] = 0; // !
// Üst kenar pikseli için 1 değeri atanır
                            }
                            else if (isBottomEdge && r == m_dim - 1) {
                                current_region[r * m_dim + c] = 0; //
Alt kenar pikseli için 1 değeri atanır
                            }
                            else {
                                // İndis geçerli değil, dışarıda kalan
bölgeler için isteğe bağlı işlemler yapılabilir
                                // Örneğin, -1 veya farklı bir değer
atanabilir
                                current_region[r * m_dim + c] = 0;
                            }
                        }
                    }
                }

                for (int i = 0; i < m_size; i++)
                { // yapısal elaman or image hedef bölgesi
                    ors_result[i] = int(current_region[i]) | int(dilation_matris[i]);
                }
                for (int i = 0; i < m_size; i++)
                { // dilation sonucu değer
                    or_result = or_result | ors_result[i];
                }

                // eğer dilation sonucu 0 ise hedef pikselin değeri azaltılır
                if (or_result == 0)
                {
                    new_data[row * im_column + col] = 0; // !
                }
                else
                {
                    new_data[row * im_column + col] = 255;
                }
                // değilse aynı kalır
            }
        }
    }

    image dilation_image;
    dilation_image.h = im_row;
    dilation_image.w = im_column;
    dilation_image.c = im.c;
    dilation_image.data = new unsigned char[image_size];

    for (int i = 0; i < image_size; i++)
    {
        dilation_image.data[i] = new_data[i];
    }

    delete[] im_data;
    delete[] current_region;
    delete[] dilation_matris;
    delete[] ors_result;
    delete[] new_data;

    return dilation_image;
}
```

```
void connectedComponent(image im, int row, int column, unsigned char label) {
        if (row < 0 || column < 0 || row >= im.h || column >= im.w) {
                return;
        }

        if (im.data[row * im.w + column] != 255) {
                return;
        }

        im.data[row * im.w + column] = label;

        connectedComponent(im, row - 1, column, label);
        connectedComponent(im, row + 1, column, label);
        connectedComponent(im, row, column - 1, label);
        connectedComponent(im, row, column + 1, label);
}

image& labeling(image& im)
{
        int row = im.h;
        int column = im.w;

        int label = 20;



        for (int r = 0; r < row; r++) {
                for (int c = 0; c < column; c++) {
                        if (im.data[r*column+c] == 255) {
                                connectedComponent(im, r, c, label);
                                label = label+10;
                }
            }
        }

        return im;
}



// çalışmadı
image& regionFilling(image& im) {
        int row = im.h;
        int column = im.w;

        unsigned char label = 255;

        image filled_image;
        filled_image.h = row;
        filled_image.w = column;
        filled_image.c = 1;

        filled_image.data = new unsigned char[row * column]{ 0 };

        for (int r = 0; r < row; r++) {
            for (int c = 0; c < column; c++) {

                // imajdaki beyaz noktalar
                if (im.data[r * column + c] == 255)
                {

                    // center
                    if (!(r < 0 || c < 0 || r >= row || c >= column)) {
                        filled_image.data[r * column + c] = label;
                    }
                    // top
                    if (!(r - 1 < 0 || c < 0 || r - 1 >= row || c >= column)) {
                        filled_image.data[(r - 1) * column + c] = label;
                    }
                    // bottom
                    if (!(r + 1 < 0 || c < 0 || r + 1 >= row || c >= column)) {
                        filled_image.data[(r + 1) * column + c] = label;
                    }
                    // left
                    if (!(r < 0 || c - 1 < 0 || r >= row || c - 1 >= column)) {
                        filled_image.data[r * column + (c - 1)] = label;
                    }
                    // right
                    if (!(r < 0 || c + 1 < 0 || r >= row || c + 1 >= column)) {
                        filled_image.data[r * column + (c + 1)] = label;
                    }

                }
            }
        }

        return filled_image;
}
```

```
// Form1.h

#pragma once
#include <atlstr.h>
#include <iostream>
#include "image.h"
#include "clustering.h"
#include "morphology.h"

namespace read_image {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Summary for Form1
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
			this->WindowState = FormWindowState::Maximized;
			//
			//TODO: Add the constructor code here
			//
```

```
    }

protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~Form1()
    {
        if (components)
        {
            delete components;
        }
    }
private: System::Windows::Forms::MenuStrip^ menuStrip1;
protected:
private: System::Windows::Forms::ToolStripMenuItem^ fileToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^ openToolStripMenuItem;
private: System::Windows::Forms::PictureBox^ pictureBox1;
private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;
private: System::Windows::Forms::ToolStripMenuItem^ clusteringToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^ histogramToolStripMenuItem;

private:
    /// <summary>
    /// Required designer variable.
    unsigned char* im_data = NULL;
    int im_w, im_h, im_c;

    unsigned char* binary_data = NULL;
    int binary_w, binary_h, binary_c;
```

```
		unsigned char* morphology_data = NULL;
		int morphology_w, morphology, morphology_c;




	private: System::Windows::Forms::DataVisualization::Charting::Chart^ histogram_chart;
	private: System::Windows::Forms::DataVisualization::Charting::Chart^ Kmeans;

	private: System::Windows::Forms::Label^ label1;
	private: System::Windows::Forms::ToolStripMenuItem^ morphologyToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ openingToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ closingToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ kmeansSegmantationToolStripMenuItem;
	private: System::Windows::Forms::PictureBox^ pictureBox2;
	private: System::Windows::Forms::ToolStripMenuItem^ regionFillingToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ labeToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ boundingToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ labelingToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ regionFillingToolStripMenuItem1;
	private: System::Windows::Forms::ToolStripMenuItem^ erosionToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^ dilationToolStripMenuItem;
		   /// </summary>
		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		void ShowRGBImages(System::Windows::Forms::PictureBox^ box, image im) {
        box->Width = 700;//im.w;
        box->Height = 750;//im.h;
        box->Refresh();
        Bitmap^ surface = gcnew Bitmap(im.w, im.h);
        box->Image = surface;
        Color c; // default de i ken
        int psw, bufpos;
        psw = im.w * im.c; // rgb ise 3 kez d ner, grey ise 1 kez
        for (int row = 0; row < im.h; row++)
            for (int col = 0; col < im.w; col++){
                bufpos = row * psw + col * im.c;
                c = Color::FromArgb(im.data[bufpos], im.data[bufpos+1], im.data[bufpos+2]); // RGB
                surface->SetPixel(col, row, c);
            }
    }//ShowImages
    void ShowIntensity(System::Windows::Forms::PictureBox^ box, image im)
    {
        box->Width = 700; // im.w;
        box->Height = 750; // im.h;
        box->Refresh();
        Bitmap^ surface = gcnew Bitmap(im.w, im.h);
        box->Image = surface;
        Color c;
        int psw, bufpos;
        psw = im.w * im.c;
        for (int row=0; row<im.h;row++)
            for (int col = 0; col < im.w; col++)
            {
                bufpos = row * psw + col * im.c;
                c = Color::FromArgb(im.data[bufpos], im.data[bufpos], im.data[bufpos]);
                surface->SetPixel(col, row, c);
            }
    }//ShowIntensity

    void ShowBinary(System::Windows::Forms::PictureBox^ box, image im)
    {
        box->Width = 700; // im.w;
        box->Height = 750; // im.h;
        box->Refresh();
        Bitmap^ surface = gcnew Bitmap(im.w, im.h);
        box->Image = surface;
        Color c;
        int psw, bufpos;
        psw = im.w * im.c;
        for (int row = 0; row < im.h; row++)
            for (int col = 0; col < im.w; col++)
            {
                bufpos = row * psw + col * im.c;
                c = Color::FromArgb(im.data[bufpos], im.data[bufpos], im.data[bufpos]);
                surface->SetPixel(col, row, c);
            }
    }//ShowIntensity

    void InitializeComponent(void)
    {
        System::Windows::Forms::DataVisualization::Charting::ChartArea^ chartArea1 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());
        System::Windows::Forms::DataVisualization::Charting::Legend^ legend1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());
        System::Windows::Forms::DataVisualization::Charting::Series^ series1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());
```

```
                    System::Windows::Forms::DataVisualization::Charting::ChartArea^
chartArea2 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());

                    System::Windows::Forms::DataVisualization::Charting::Legend^ legend2 =
(gcnew System::Windows::Forms::DataVisualization::Charting::Legend());

                    System::Windows::Forms::DataVisualization::Charting::Series^ series2 =
(gcnew System::Windows::Forms::DataVisualization::Charting::Series());

                    this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());

                    this->fileToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->openToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->clusteringToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->histogramToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->kmeansSegmantationToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->morphologyToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->openingToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->closingToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->regionFillingToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->regionFillingToolStripMenuItem1 = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->erosionToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->dilationToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->labeToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->boundingToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());

                    this->labelingToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
```

```
			this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
			this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());
			this->histogram_chart = (gcnew System::Windows::Forms::DataVisualization::Charting::Chart());
			this->Kmeans = (gcnew System::Windows::Forms::DataVisualization::Charting::Chart());
			this->label1 = (gcnew System::Windows::Forms::Label());
			this->pictureBox2 = (gcnew System::Windows::Forms::PictureBox());
			this->menuStrip1->SuspendLayout();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->histogram_chart))->BeginInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->Kmeans))->BeginInit();
			(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox2))->BeginInit();
			this->SuspendLayout();
			//
			// menuStrip1
			//
			this->menuStrip1->ImageScalingSize = System::Drawing::Size(20, 20);
			this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^  >(4) {
				this->fileToolStripMenuItem,
					this->clusteringToolStripMenuItem, this->morphologyToolStripMenuItem, this->labeToolStripMenuItem
			});
			this->menuStrip1->Location = System::Drawing::Point(0, 0);
			this->menuStrip1->Name = L"menuStrip1";
			this->menuStrip1->Padding = System::Windows::Forms::Padding(5, 2, 0, 2);
			this->menuStrip1->Size = System::Drawing::Size(1902, 28);
			this->menuStrip1->TabIndex = 0;
			this->menuStrip1->Text = L"menuStrip1";
```

```
            //
            // fileToolStripMenuItem
            //
            this->fileToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^  >(1) { this->openToolStripMenuItem });
            this->fileToolStripMenuItem->Name = L"fileToolStripMenuItem";
            this->fileToolStripMenuItem->Size = System::Drawing::Size(46, 24);
            this->fileToolStripMenuItem->Text = L"File";
            //
            // openToolStripMenuItem
            //
            this->openToolStripMenuItem->Name = L"openToolStripMenuItem";
            this->openToolStripMenuItem->Size = System::Drawing::Size(128, 26);
            this->openToolStripMenuItem->Text = L"Open";
            this->openToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::openToolStripMenuItem_Click);
            //
            // clusteringToolStripMenuItem
            //
            this->clusteringToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^  >(2) {
                this->histogramToolStripMenuItem,
                    this->kmeansSegmantationToolStripMenuItem
            });
            this->clusteringToolStripMenuItem->Name = L"clusteringToolStripMenuItem";
            this->clusteringToolStripMenuItem->Size = System::Drawing::Size(89, 24);
            this->clusteringToolStripMenuItem->Text = L"Clustering";
            //
            // histogramToolStripMenuItem
            //
            this->histogramToolStripMenuItem->Name = L"histogramToolStripMenuItem";
```

```
			this->histogramToolStripMenuItem->Size = System::Drawing::Size(249, 26);
			this->histogramToolStripMenuItem->Text = L"Histogram_Extraction";
			this->histogramToolStripMenuItem->Click += gcnew
System::EventHandler(this, &Form1::histogramToolStripMenuItem_Click);
			// 
			// kmeansSegmantationToolStripMenuItem
			// 
			this->kmeansSegmantationToolStripMenuItem->Name =
L"kmeansSegmantationToolStripMenuItem";
			this->kmeansSegmantationToolStripMenuItem->Size =
System::Drawing::Size(249, 26);
			this->kmeansSegmantationToolStripMenuItem->Text = L"K-
means_Segmantation";
			this->kmeansSegmantationToolStripMenuItem->Click += gcnew
System::EventHandler(this, &Form1::kmeansSegmantationToolStripMenuItem_Click);
			// 
			// morphologyToolStripMenuItem
			// 
			this->morphologyToolStripMenuItem->DropDownItems->AddRange(gcnew
cli::array< System::Windows::Forms::ToolStripItem^  >(6) {
				this->openingToolStripMenuItem,
					this->closingToolStripMenuItem, this-
>regionFillingToolStripMenuItem, this->regionFillingToolStripMenuItem1, this-
>erosionToolStripMenuItem,
					this->dilationToolStripMenuItem
			});
			this->morphologyToolStripMenuItem->Name =
L"morphologyToolStripMenuItem";
			this->morphologyToolStripMenuItem->Size = System::Drawing::Size(105, 24);
			this->morphologyToolStripMenuItem->Text = L"Morphology";
			// 
			// openingToolStripMenuItem
			// 
			this->openingToolStripMenuItem->Name = L"openingToolStripMenuItem";
			this->openingToolStripMenuItem->Size = System::Drawing::Size(224, 26);
			this->openingToolStripMenuItem->Text = L"Opening";
			this->openingToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::openingToolStripMenuItem_Click);
			// 
			// closingToolStripMenuItem
			// 
			this->closingToolStripMenuItem->Name = L"closingToolStripMenuItem";
			this->closingToolStripMenuItem->Size = System::Drawing::Size(224, 26);
			this->closingToolStripMenuItem->Text = L"Closing";
			this->closingToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::closingToolStripMenuItem_Click);
			// 
			// regionFillingToolStripMenuItem
			// 
			this->regionFillingToolStripMenuItem->Name = L"regionFillingToolStripMenuItem";
			this->regionFillingToolStripMenuItem->Size = System::Drawing::Size(224, 26);
			this->regionFillingToolStripMenuItem->Text = L"Edge_Extraction";
			this->regionFillingToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::edgeExtractToolStripMenuItem_Click);
			// 
			// regionFillingToolStripMenuItem1
			// 
			this->regionFillingToolStripMenuItem1->Name = L"regionFillingToolStripMenuItem1";
			this->regionFillingToolStripMenuItem1->Size = System::Drawing::Size(224, 26);
			this->regionFillingToolStripMenuItem1->Text = L"Region_Filling";
			this->regionFillingToolStripMenuItem1->Click += gcnew System::EventHandler(this, &Form1::regionFillingToolStripMenuItem1_Click);
			// 
			// erosionToolStripMenuItem
			// 
```

```
            this->erosionToolStripMenuItem->Name = L"erosionToolStripMenuItem";
            this->erosionToolStripMenuItem->Size = System::Drawing::Size(224, 26);
            this->erosionToolStripMenuItem->Text = L"Erosion";
            this->erosionToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::erosionToolStripMenuItem_Click);
            //
            // dilationToolStripMenuItem
            //
            this->dilationToolStripMenuItem->Name = L"dilationToolStripMenuItem";
            this->dilationToolStripMenuItem->Size = System::Drawing::Size(224, 26);
            this->dilationToolStripMenuItem->Text = L"Dilation";
            this->dilationToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::dilationToolStripMenuItem_Click);
            //
            // labeToolStripMenuItem
            //
            this->labeToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(2) {
                this->boundingToolStripMenuItem,
                    this->labelingToolStripMenuItem
            });
            this->labeToolStripMenuItem->Name = L"labeToolStripMenuItem";
            this->labeToolStripMenuItem->Size = System::Drawing::Size(150, 24);
            this->labeToolStripMenuItem->Text = L"Labeling-Bounding";
            //
            // boundingToolStripMenuItem
            //
            this->boundingToolStripMenuItem->Name = L"boundingToolStripMenuItem";
            this->boundingToolStripMenuItem->Size = System::Drawing::Size(156, 26);
            this->boundingToolStripMenuItem->Text = L"Bounding";
            //
            // labelingToolStripMenuItem
```

```
			//
			this->labelingToolStripMenuItem->Name = L"labelingToolStripMenuItem";
			this->labelingToolStripMenuItem->Size = System::Drawing::Size(156, 26);
			this->labelingToolStripMenuItem->Text = L"Labeling";
			this->labelingToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Form1::labelingToolStripMenuItem_Click);
			//
			// pictureBox1
			//
			this->pictureBox1->Location = System::Drawing::Point(15, 75);
			this->pictureBox1->Margin = System::Windows::Forms::Padding(4);
			this->pictureBox1->Name = L"pictureBox1";
			this->pictureBox1->Size = System::Drawing::Size(650, 750);
			this->pictureBox1->TabIndex = 1;
			this->pictureBox1->TabStop = false;
			//
			// openFileDialog1
			//
			this->openFileDialog1->FileName = L"openFileDialog1";
			//
			// histogram_chart
			//
			chartArea1->Name = L"ChartArea1";
			this->histogram_chart->ChartAreas->Add(chartArea1);
			legend1->Name = L"Legend1";
			this->histogram_chart->Legends->Add(legend1);
			this->histogram_chart->Location = System::Drawing::Point(1343, 75);
			this->histogram_chart->Margin = System::Windows::Forms::Padding(3, 2, 3,
2);
			this->histogram_chart->Name = L"histogram_chart";
			series1->ChartArea = L"ChartArea1";
```

```
                    series1->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::FastLine;
                    series1->Legend = L"Legend1";
                    series1->Name = L"Histogram";
                    this->histogram_chart->Series->Add(series1);
                    this->histogram_chart->Size = System::Drawing::Size(435, 255);
                    this->histogram_chart->TabIndex = 2;
                    this->histogram_chart->Text = L"chart1";
                    this->histogram_chart->Visible = false;
                    //
                    // Kmeans
                    //
                    chartArea2->Name = L"ChartArea1";
                    this->Kmeans->ChartAreas->Add(chartArea2);
                    legend2->Name = L"Legend1";
                    this->Kmeans->Legends->Add(legend2);
                    this->Kmeans->Location = System::Drawing::Point(1448, 339);
                    this->Kmeans->Margin = System::Windows::Forms::Padding(4);
                    this->Kmeans->Name = L"Kmeans";
                    series2->ChartArea = L"ChartArea1";
                    series2->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Point;
                    series2->Legend = L"Legend1";
                    series2->Name = L"Kmeans";
                    series2->YValuesPerPoint = 2;
                    this->Kmeans->Series->Add(series2);
                    this->Kmeans->Size = System::Drawing::Size(429, 257);
                    this->Kmeans->TabIndex = 3;
                    this->Kmeans->Text = L"Kmeans";
                    this->Kmeans->Visible = false;
                    //
                    // label1
```

```
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(21, 36);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(70, 16);
this->label1->TabIndex = 4;
this->label1->Text = L"Message: ";
//
// pictureBox2
//
this->pictureBox2->Location = System::Drawing::Point(672, 75);
this->pictureBox2->Name = L"pictureBox2";
this->pictureBox2->Size = System::Drawing::Size(650, 750);
this->pictureBox2->TabIndex = 1;
this->pictureBox2->TabStop = false;
//
// Form1
//
this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(1902, 1033);
this->Controls->Add(this->pictureBox2);
this->Controls->Add(this->label1);
this->Controls->Add(this->Kmeans);
this->Controls->Add(this->histogram_chart);
this->Controls->Add(this->pictureBox1);
this->Controls->Add(this->menuStrip1);
this->MainMenuStrip = this->menuStrip1;
this->Margin = System::Windows::Forms::Padding(4);
this->Name = L"Form1";
this->Text = L"Form1";
```

```
                              this->menuStrip1->ResumeLayout(false);
                              this->menuStrip1->PerformLayout();
                              (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>pictureBox1))->EndInit();
                              (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>histogram_chart))->EndInit();
                              (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>Kmeans))->EndInit();
                              (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>pictureBox2))->EndInit();
                              this->ResumeLayout(false);
                              this->PerformLayout();


                    }
#pragma endregion
          private: System::Void openToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
                    CString str;


                    if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
                              //pictureBox1->ImageLocation = openFileDialog1->FileName;
                              str = openFileDialog1->FileName;
                              CStringA s2(str);
                              const char* input = s2;
                              image im = load_image(input);
                              ShowRGBImages(pictureBox1, im);


                              // shallow copy
                              im_data = im.data;
                              im_h = im.h;
                              im_w = im.w;
                              im_c = im.c;
```

```
            label1->Text = L"Message: Image was picked and have been showing in RGB
mode.";


            std::cout <<"w: " << im.w<<"\n";

            std::cout << "h: " << im.h << "\n";

            std::cout <<"c: " << im.c << "\n";

            std::cout << "data[10]: " << (int)im.data[10] << "\n";

        }//


    }//openTool




    private: System::Void histogramToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    // RGB to Intensity


    if (im_data == NULL) {
        MessageBox::Show("Okunacak Image  ncelikle se ilmeli!");
        }
    else {
        image im;
        im.w = im_w;
        im.h = im_h;
        im.c = im_c;
        im.data = im_data;


        image gray_im = RGBtoIntensity(im);


        int* hist_data = Histogram(gray_im);


        //raw_data = gray_im.data;
        ShowIntensity(pictureBox1, gray_im);
        histogram_chart->Visible = true;

        histogram_chart->Series["Histogram"]->Points->Clear();


        histogram_chart->Location = System::Drawing::Point(pictureBox1->Width+500, 75);
//1225

        for (int i = 0; i < 256; i++) { // histogram 256 elemanl

            histogram_chart->Series["Histogram"]->Points->AddXY(i, hist_data[i]);

        }

        label1->Text = L"Message: Image was turned into Gray-Level mode and its intensty
value histogram graph has been extract.";

    }

}//histogram_extraction func




    private: System::Void kmeansSegmantationToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {

        if (im_data == NULL) {

            MessageBox::Show("Okunacak Image  ncelikle se ilmeli!");

        }

        else {

            // rgb resmi al

            image im;

            im.w = im_w;

            im.h = im_h;

            im.c = im_c;

            im.data = im_data;




            // gray level'a  evir

            image gray_im = RGBtoIntensity(im);
            // kmeans de erlerini bul
            float* means = new float[2];
            means[0] = 0.0;
            means[1] = 0.0;
            means = KMeans_Euclidean(gray_im, 2);

            // kmeans ile segmentasyon yap ve binary image'i elde et
            image binary_im;
            binary_im = KBasedSegmentation(gray_im, means, 2);

            // binary image'i g ster
            ShowBinary(pictureBox2, binary_im);

            binary_data = binary_im.data;
            binary_h = binary_im.h;
            binary_w = binary_im.w;
            binary_c = binary_im.c;

            label1->Text = L"Message: Image in binary mode";
        }
    }


    private: System::Void openingToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e)

    {
        image im;
        im.data = binary_data;
        im.c = binary_c;
        im.h = binary_h;
```

```
			im.w = binary_w;

			im = opening(im,3);



			ShowBinary(pictureBox2, im);

			binary_data = im.data;
			binary_h = im.h;
			binary_w = im.w;
			binary_c = im.c;



			label1->Text = L"Message: Image in binary mode after opening.";

		}



		private: System::Void closingToolStripMenuItem_Click(System::Object^ sender, 
System::EventArgs^ e) {

		image im;
		im.data = binary_data;
		im.c = binary_c;
		im.h = binary_h;
		im.w = binary_w;

		im = closing(im,3);



		ShowBinary(pictureBox2, im);
```

```
    binary_data = im.data;
    binary_h = im.h;
    binary_w = im.w;
    binary_c = im.c;

    label1->Text = L"Message: Image in binary mode after closing.";

}


    private: System::Void edgeExtractToolStripMenuItem_Click(System::Object^ sender, 
System::EventArgs^ e) {

        image im;
        im.data = binary_data;
        im.c = binary_c;
        im.h = binary_h;
        im.w = binary_w;

        im = edge_detection(im, 3);

        ShowIntensity(pictureBox2, im);

        binary_data = im.data;
        binary_h = im.h;
        binary_w = im.w;
        binary_c = im.c;


        label1->Text = L"Message: Image in binary mode after edge detection.";
```

```
	}



	private: System::Void labelingToolStripMenuItem_Click(System::Object^ sender, 
System::EventArgs^ e) {

		image im;
		im.data = binary_data;
		im.c = binary_c;
		im.h = binary_h;
		im.w = binary_w;

		im = labeling(im);

		im = Intensity2RGB(im);

		ShowRGBImages(pictureBox2, im);

		binary_data = im.data;
		binary_h = im.h;
		binary_w = im.w;
		binary_c = im.c;



		label1->Text = L"Message: Image in binary mode after labeling.";

	}



	private: System::Void regionFillingToolStripMenuItem1_Click(System::Object^ sender, 
System::EventArgs^ e) {
		image im;
		im.data = binary_data;
		im.c = binary_c;
		im.h = binary_h;
		im.w = binary_w;

		im = regionFilling(im);

		ShowBinary(pictureBox2, im);

		binary_data = im.data;
		binary_h = im.h;
		binary_w = im.w;
		binary_c = im.c;


		label1->Text = L"Message: Image in binary mode after region filling.";
	}


	private: System::Void erosionToolStripMenuItem_Click(System::Object^ sender, 
System::EventArgs^ e) {
		image im;
		im.data = binary_data;
		im.c = binary_c;
		im.h = binary_h;
		im.w = binary_w;

		im = erosion(im,3);

		ShowBinary(pictureBox2, im);
```

```
        binary_data = im.data;
        binary_h = im.h;
        binary_w = im.w;
        binary_c = im.c;



        label1->Text = L"Message: Image in binary mode after erosion.";
    }



    private: System::Void dilationToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
        image im;
        im.data = binary_data;
        im.c = binary_c;
        im.h = binary_h;
        im.w = binary_w;

        im = dilation(im,3);

        ShowBinary(pictureBox2, im);

        binary_data = im.data;
        binary_h = im.h;
        binary_w = im.w;
        binary_c = im.c;



        label1->Text = L"Message: Image in binary mode after dilation.";
    }
```

```
};

}




// ************************************************************************************


// Form1.cpp

#include "Form1.h"
```

```
#include "Form1.h";

using namespace read_image;

[STAThread]
int main(array<System::String^>^ args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Application::Run(gcnew Form1());
    return 0;
}
```