```
// 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;
}
```