```cpp
//Mohan Karthik Ramaraj

// Homework.cpp : Defines the entry point for the console application.
//

#include <afxwin.h>  // necessary for MFC to work properly
#include "Homework.h"
#include<iostream>
#include<stdio.h>
#include "../../src/blepo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

using namespace blepo;

int main(int argc, const char* argv[])
{
    // Initialize MFC and return if failure
    HMODULE hModule = ::GetModuleHandle(NULL);
    if (hModule == NULL || !AfxWinInit(hModule, NULL, ::GetCommandLine(), 0))
    {
        printf("Fatal Error: MFC initialization failed (hModule = %x)\n", hModule);
        return 1;
    }

    try
    {
        //Checking number of arguements
        if (argc<2)
            printf("Command Line Parameters Not specified");
        else
        {

            char str[80];
            strcpy(str, "../../images/");
            strcat(str,argv[1]);

            // Load filename1.
            ImgGray img;
            Load(str, &img);



            // Display image in figure window.
            Figure fig(("Original Image"),0,0,true);
            fig.Draw(img);
            int iw=img.Width(),ih=img.Height();

```

```
            //Checking Image Dimension
            if (ih<=0||iw<=0)
                printf("Dimensions Incorrect");
            else
            {
                //Low Threshold Image
                int al=80;                                  //Low Threshold Value
                ImgBinary OutputImg;
                Threshold(img, al, &OutputImg);             //Thresholding
                ImgBinary LTI;
                Erode3x3(OutputImg, &LTI);                  //Eroding Image
                Figure fig2(("Low Threshold"),300,0,true);  //Display LTI
                fig2.Draw(LTI);


                //High Threshold
                int ah=150;                                 //High Threshold Value
                ImgBinary OutputImg1;
                Threshold(img, ah, &OutputImg1);            //Thresholding
                ImgBinary HTI;
                Erode3x3(OutputImg1, &HTI);                 //Eroding Image
                Figure fig3(("High Threshold"),300,350,true);   //Display HTI
                fig3.Draw(HTI);

                //Double Threshold Using Floodfill
                unsigned char val;
                ImgBinary DTI1(iw,ih);
                ImgBinary DTI;
                Set(&DTI1,0);
                for (int y=0;y<ih;y++)
                {
                    for(int x=0;x<iw;x++)
                    {
                        val=HTI(x,y);
                        if (val==1)
                            FloodFill4(LTI,x,y,val,&DTI1);  //Floodfill

                    }
                }
                Erode3x3(DTI1,&DTI);                        //Eroding Image
                Figure fig4(("Double Threshold"),600,0,true);   //Display DTI
                fig4.Draw(DTI);

                //Connected Components
                ImgInt labels;
                int reg = ConnectedComponents4(DTI, &labels);
                int creg=reg-1;                             //Counting Fruits
                printf("The image has %d fruits",creg);
                ImgGray lbl;
                Convert(labels, &lbl);
```

```
101.
102.                 //Multiply
103.                 ImgInt out;
104.                 ImgInt mult(iw,ih);
105.                 Set(&mult,30);
106.                 Multiply(labels, mult, &out);
107.                 ImgGray out1;
108.                 Convert(out, &out1);
109.                 Figure fig6(("Connected Components"),900,0,true);           //Display Connected
    Component Image
110.                 fig6.Draw(out1);
111.
112.                 //Segmenting Regions
113.                 int count=0;
114.                 ImgBgr OrigImg;
115.                 Load(str,&OrigImg);
116.                 ImgBinary peri;
117.                 ImgBinary ero;
118.                 ImgBinary singreg(iw,ih);
119.                 Set(&singreg, 0);
120.                 for(int i = 1; i < reg; i++)
121.                 {
122.                     Set(&singreg, 0);
123.                     for(int y = 0; y <ih; y++)
124.                     {
125.                         for(int x = 0; x < iw; x++)
126.                         {
127.                             if(labels(x, y) == i)
128.                             {
129.                                 singreg(x, y) = 1;
130.
131.                             }
132.                         }
133.                     }
134.
135.                     //Getting Outline
136.                     Erode3x3(singreg, &peri);
137.                     Xor(singreg, peri, &ero);
138.
139.                     //Calculating Perimeter
140.                     for (int y=0;y<ih;y++)
141.                     {
142.                         for(int x=0;x<iw;x++)
143.                         {
144.
145.                             val=ero(x,y);
146.                             if (val==1)
147.                                 count++;
148.                         }
149.                     }
```

```
150.
151.
152.                     RegionProperties Rps;
153.                     RegionProps(singreg, &Rps);
154.
155.                     //Calculating Major and Minor Axes Lengths
156.                     double bda1 = 1/(2*Rps.mu00) * ( Rps.mu20 + Rps.mu02 + abs(sqrt (
    pow(Rps.mu20 - Rps.mu02,2) + pow(2*Rps.mu11,2) )) );
157.                     double bda2 = 1/(2*Rps.mu00) * ( Rps.mu20 + Rps.mu02 - abs(sqrt (
    pow(Rps.mu20 - Rps.mu02,2) + pow(2*Rps.mu11,2) ) ));
158.                     int majlength = abs(sqrt(bda1));
159.                     int minlength = abs(sqrt(bda2));
160.                     double vector1 = pow(majlength, 2);
161.                     double vector2 = pow(minlength, 2);
162.
163.                     //Drawing Cross
164.                     Point cross1=Point(int(Rps.xc+3), int(Rps.yc));
165.                     Point cross2=Point(int(Rps.xc-3), int(Rps.yc));
166.                     Point cross3=Point(int(Rps.xc), int(Rps.yc+3));
167.                     Point cross4=Point(int(Rps.xc), int(Rps.yc-3));
168.                     DrawLine(cross1,cross2,&OrigImg,Bgr(255,0,0));
169.                     DrawLine(cross3,cross4,&OrigImg,Bgr(255,0,0));
170.
171.                     //Drawing Axes
172.                     double axm=majlength*cos(Rps.direction);
173.                     double axm1=majlength*sin(Rps.direction);
174.                     double axn=minlength*cos(Rps.direction);
175.                     double axn1=minlength*sin(Rps.direction);
176.
177.                     Point x1=Point(int(Rps.xc+axm), int(Rps.yc+axm1));
178.                     Point y1=Point(int(Rps.xc-axm), int(Rps.yc-axm1));
179.                     Point x2=Point(int(Rps.xc-axn1), int(Rps.yc+axn));
180.                     Point y2=Point(int(Rps.xc+axn1), int(Rps.yc-axn));
181.                     DrawLine(x1,y1,&OrigImg,Bgr(255,0,0));
182.                     DrawLine(x2,y2,&OrigImg,Bgr(255,0,0));
183.
184.                     //Classify and Print Properties
185.                     if (Rps.eccentricity>0.8)                    //Banana
186.                     {
187.                         double C=(4*3.14*Rps.area)/(count*count);   //Compactness
188.                         printf("\n\nBanana\n Eccentricity is:%lf\n Regular 0th Order is:%lf\n
    Regular 1st Order is:%lf\n Regular 2nd Order is:%lf\n Centralized 0th Order is:%lf\n Centralized
    1st Order is:%lf\n Centralized 2nd Order is:%lf\n Direction is:%lf\n Compactness
    is:%f",Rps.eccentricity,Rps.m00,Rps.m01,Rps.m02,Rps.mu00,Rps.mu01,Rps.mu02,Rps.direction,C);
189.
190.                         //Finding the Banana Stem
191.                         ImgBinary temp;
192.                         ImgBinary temp1;
193.                         ImgBinary temp2;
194.                         ImgBinary temp3;
```

```
                        ImgBinary temp4;
                        ImgBinary temp5;
                        ImgBinary temp6;
                        ImgBinary stem;
                        Erode3x3(singreg, &temp);
                        Erode3x3(temp, &temp1);
                        Erode3x3(temp1, &temp2);
                        Dilate3x3(temp2, &temp3);
                        Dilate3x3(temp3,&temp4);
                        Dilate3x3(temp4,&temp5);
                        Or(ero,temp5,&temp6);
                        Xor(temp6, temp5, &stem);

                        //Colouring the Boundary
                        for (int y=0;y<ih;y++)
                        {
                            for(int x=0;x<iw;x++)
                            {

                                val=ero(x,y);
                                if (val==1)
                                    OrigImg(x,y)=Bgr(0,255,255);
                                unsigned char val1=(x,y);
                                val1=stem(x,y);
                                if(val1==1)
                                    OrigImg(x,y)=Bgr(255,0,255);
                            }
                        }

                    }


                    else if (Rps.eccentricity<0.5&&Rps.area>5000)//GrapeFruit
                    {
                        double C=(4*3.14*Rps.area)/(count*count);//Compactness
                        printf("\n\nGrapefruit\n Eccentricity is:%lf\n Regular 0th Order is:%lf\n Regular 1st Order is:%lf\n Regular 2nd Order is:%lf\n Centralized 0th Order is:%lf\n Centralized 1st Order is:%lf\n Centralized 2nd Order is:%lf\n Direction is:%lf\n Compactness is:%f",Rps.eccentricity,Rps.m00,Rps.m01,Rps.m02,Rps.mu00,Rps.mu01,Rps.mu02,Rps.direction,C);


                        for (int y=0;y<ih;y++)                    //Colouring the Boundary
                        {
                            for(int x=0;x<iw;x++)
                            {

                                val=ero(x,y);
                                if (val==1)
                                    OrigImg(x,y)=Bgr(0,255,0);
                            }
```

```
                        }
                    }


                    else if (Rps.eccentricity<0.5&&Rps.area<4500)//Apple
                    {
                        double C=(4*3.14*Rps.area)/(count*count);//Compactness
                        printf("\n\nApple\n Eccentricity is:%lf\n Regular 0th Order is:%lf\n Regular 1st Order is:%lf\n Regular 2nd Order is:%lf\n Centralized 0th Order is:%lf\n Centralized 1st Order is:%lf\n Centralized 2nd Order is:%lf\n Direction is:%lf\n Compactness is:%f",Rps.eccentricity,Rps.m00,Rps.m01,Rps.m02,Rps.mu00,Rps.mu01,Rps.mu02,Rps.direction,C);

                        for (int y=0;y<ih;y++)                  //Colouring the Boundary
                        {
                            for(int x=0;x<iw;x++)
                            {
                                val=ero(x,y);
                                if (val==1)
                                    OrigImg(x,y)=Bgr(0,0,255);
                            }
                        }
                    }
                    count=0;
                }
                //Displaying the Final Image
                Figure fig9(("Final Image"),900,350,true);
                fig9.Draw(OrigImg);
            }
        }
        // Loop forever until user presses Ctrl-C in terminal window.
        EventLoop();
    }

    catch (const Exception& e)
    {
        e.Display();    // display exception to user in a popup window
    }

    return 0;
}
```