```c
/*
    Name: Rodrigo Ignacio Rojas Garcia
    Lab#: 4
*/

// Library Declaration Section
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <sys/timeb.h>
#include <windows.h>
#include <wingdi.h>
#include <winuser.h>
#include <process.h>
#include "resource.h"
#include "globals.h"

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine,
int nCmdShow)
{
    MSG         msg;
    HWND        hWnd;
    WNDCLASS    wc;

    wc.style=CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc=(WNDPROC)WndProc;
    wc.cbClsExtra=0;
    wc.cbWndExtra=0;
    wc.hInstance=hInstance;
    wc.hIcon=LoadIcon(hInstance,"ID_PLUS_ICON");
    wc.hCursor=LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName="ID_MAIN_MENU";
    wc.lpszClassName="PLUS";

    if (!RegisterClass(&wc))
    {
        return(FALSE);
    }

    hWnd=CreateWindow("PLUS","plus program", WS_OVERLAPPEDWINDOW | WS_HSCROLL |
    WS_VSCROLL, CW_USEDEFAULT,0,400,400,NULL,NULL,hInstance,NULL);
    if (!hWnd)
    {
        return(FALSE);
    }

    ShowScrollBar(hWnd,SB_BOTH,FALSE);
    ShowWindow(hWnd,nCmdShow);
    UpdateWindow(hWnd);
    MainWnd=hWnd;

    /* SETS GLOBAL VARIABLES ONCE PROGRAM STARTS */
    ShowPixelCoords=0;
    play_mode = 0;
    step_mode = 0;
    total_threads = 0;

    strcpy(filename,"");
    OriginalImage=NULL;
    ROWS=COLS=0;

    InvalidateRect(hWnd,NULL,TRUE);
    UpdateWindow(hWnd);

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return(msg.wParam);
}

/* CALLBACK FOR PIXEL INTENSITY*/
LRESULT CALLBACK WndProc2(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
            case IDOK:
                GetDlgItemText(hWnd, IDC_EDIT1, threshold, 256);
                thresh = atoi(threshold);
                EndDialog(hWnd, wParam);
                break;
            case IDCANCEL:
                EndDialog(hWnd, wParam);
        }
    }
    return(0L);
}

/* CALLBACK FOR CENTROID DISNTACE */
LRESULT CALLBACK WndProc3(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
        case IDOK:
            GetDlgItemText(hWnd, IDC_EDIT1, radius, 256);
            rad = atoi(radius);
            EndDialog(hWnd, wParam);
            break;
        case IDCANCEL:
            EndDialog(hWnd, wParam);
        }
    }
    return(0L);
}

LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    // Variable Declaration Section
    HMENU               hMenu;
    OPENFILENAME        ofn;
    FILE                *fpt;
    HDC                 hDC;
    char                header[320],text[320];
    int                 BYTES,xPos,yPos;

    switch (uMsg)
    {
        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                /* EVENT WHEN USER SELECT SHOW PIXEL */
                case ID_SHOWPIXELCOORDS:
                    ShowPixelCoords = (ShowPixelCoords + 1) % 2;
                    //PaintImage();
                    break;

                /* EVENT WHEN USER SELECTS PLAY MODE*/
                case ID_PLAY_MODE:
                    play_mode = (play_mode + 1) % 2;
```

```c
                step_mode = 0;
                end_thread = 0;
                break;

            /* EVENT WHEN USER SELECTS STEP MODE*/
            case ID_STEP_MODE:
                step_mode = (step_mode + 1) % 2;
                play_mode = 0;
                end_thread = 0;
                break;

            /* CREATES DIALOG BOX FOR COLORS*/
            case ID_SELECT_COLOR:
                ZeroMemory(&color, sizeof(CHOOSECOLOR));
                color.lStructSize = sizeof(CHOOSECOLOR);
                color.hwndOwner = hWnd;
                color.lpCustColors = (LPDWORD)acrCustClr;
                color.rgbResult = rgbCurrent;
                color.Flags = CC_FULLOPEN | CC_RGBINIT;
                if (ChooseColor(&color) == TRUE)
                {
                    hbrush = CreateSolidBrush(color.rgbResult);
                    rgbCurrent = color.rgbResult;
                }
                break;

            /* EVENT WHEN USER SELECTS UNDO */
            case ID_UNDO:
                end_thread = 1;
                int i, j;
                HDC hDC;
                /* UNDO'S LAST THREAD DRAWING BY SETTING PIXELS CHANGED TO ORIGINAL IMAGE RGB VALUES*/
                for (i = 0; i < ROWS; i++)
                {
                    for (j = 0; j < COLS; j++)
                    {
                        if (indices[i * COLS + j] == total_threads)
                        {
                            hDC = GetDC(MainWnd);
                            SetPixel(hDC, j, i, RGB(OriginalImage[i * COLS + j],
                                OriginalImage[i * COLS + j], OriginalImage[i * COLS + j]));
                            ReleaseDC(MainWnd, hDC);
                            indices[i * COLS + j] = 0;
                        }
                    }
                }
                end_thread = 0;
                if (total_threads > 0)
                {
                    total_threads -= 1;
                }
                break;

            /* EVENT WHEN USER SELECTS TO CHANGE PIXEL INTENSITY*/
            case ID_PIXEL_INTENSITY:
                DialogBox(NULL, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, WndProc2);
                break;

            /* EVENT WHEN USER SELECTS TO CHANGE THE CENTROID DISTANCE */
            case ID_CENTROID_DISTANCE:
                DialogBox(NULL, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, WndProc3);
                break;

            /* EVENT WHEN USER SELECTS TO LOAD A PICTURE TO THE GUI*/
            case ID_FILE_LOAD:
                if (OriginalImage != NULL)
                {
```

```
                            free(OriginalImage);
                            OriginalImage = NULL;
                        }
                        memset(&(ofn), 0, sizeof(ofn));
                        ofn.lStructSize = sizeof(ofn);
                        ofn.lpstrFile = filename;
                        filename[0] = 0;
                        ofn.nMaxFile = MAX_FILENAME_CHARS;
                        ofn.Flags = OFN_EXPLORER | OFN_HIDEREADONLY;
                        ofn.lpstrFilter = "PPM files\0*.ppm\0All files\0*.*\0\0";
                        if (!(GetOpenFileName(&ofn)) || filename[0] == '\0')
                        {
                            break;      /* user cancelled load */
                        }
                        if ((fpt = fopen(filename, "rb")) == NULL)
                        {
                            MessageBox(NULL, "Unable to open file", filename, MB_OK |
                            MB_APPLMODAL);
                            break;
                        }
                        fscanf(fpt, "%s %d %d %d", header, &COLS, &ROWS, &BYTES);
                        if (strcmp(header, "P5") != 0 || BYTES != 255)
                        {
                            MessageBox(NULL, "Not a PPM (P5 greyscale) image", filename,
                            MB_OK | MB_APPLMODAL);
                            fclose(fpt);
                            break;
                        }
                        OriginalImage = (unsigned char *)calloc(ROWS*COLS, 1);
                        header[0] = fgetc(fpt); /* whitespace character after header */
                        fread(OriginalImage, 1, ROWS*COLS, fpt);
                        fclose(fpt);
                        SetWindowText(hWnd, filename);
                        PaintImage();

                        /* CREATES A GLOBAL VARIABLE FOR INDICES*/
                        indices = (int *)calloc(ROWS * COLS, sizeof(int));

                        break;

                    case ID_FILE_QUIT:
                        DestroyWindow(hWnd);
                        break;
                }
                break;

            case WM_SIZE:         /* could be used to detect when window size changes */
                PaintImage();
                return(DefWindowProc(hWnd,uMsg,wParam,lParam));
                break;

            case WM_PAINT:
                PaintImage();
                return(DefWindowProc(hWnd,uMsg,wParam,lParam));
                break;

            case WM_LBUTTONDOWN:case WM_RBUTTONDOWN:

                /* MOUSE CLICK FOR PLAY MODE OR STEP MODE */
                if ((play_mode == 1) || (step_mode == 1))
                {
                    /* GETS THE X AND Y POSITION OF CLICKED POSITION IN IMAGE*/
                    mouse_x_pos = LOWORD(lParam);
                    mouse_y_pos = HIWORD(lParam);

                    /* CREATES A THREAD AND BEGINS REGION GROW ON IMAGE */
                    if ((mouse_x_pos >= 0) && (mouse_x_pos < COLS) && (mouse_y_pos >= 0) &&
                    (mouse_y_pos < ROWS));
                    {
```

```c
                    fill_thread_running = 1;
                    _beginthread(region_grow, 0, MainWnd);
                    total_threads += 1;
                }

            }
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;

        /* EVENT THAT SHOWS CURRENT CURSOR PIXEL AND DRAWS ON IMAGE */
        case WM_MOUSEMOVE:
            if (ShowPixelCoords == 1)
            {
                  xPos=LOWORD(lParam);
                  yPos=HIWORD(lParam);
                  if (xPos >= 0  &&  xPos < COLS  &&  yPos >= 0  &&  yPos < ROWS)
                  {
                        sprintf(text,"%d, %d => %d", xPos, yPos, OriginalImage[yPos*COLS+xPos]);
                        hDC=GetDC(MainWnd);
                        TextOut(hDC,0,0,text,strlen(text));     /* draw text on the window */
                        SetPixel(hDC,xPos,yPos,RGB(255,0,0));   /* color the cursor position red */
                        ReleaseDC(MainWnd,hDC);
                  }
            }
        return(DefWindowProc(hWnd,uMsg,wParam,lParam));
        break;

        case WM_KEYDOWN:
            if (wParam == 's' || wParam == 'S')
            {
                PostMessage(MainWnd, WM_COMMAND, ID_SHOWPIXELCOORDS, 0);      /* send message to self */
            }
            /* VENT THAT TRIGGERS WHEN USER PRESSES J ON STEP MODE, THE REGION KEEPS GROWING */
            if (wParam == 'j' || wParam == 'J')
            {
                is_j_pressed = 1;
            }
            if ((TCHAR)wParam == '1')
            {
                TimerRow=TimerCol=0;
                SetTimer(MainWnd,TIMER_SECOND,10,NULL); /* start up 10 ms timer */
            }
            if ((TCHAR)wParam == '2')
            {
                KillTimer(MainWnd,TIMER_SECOND);            /* halt timer, stopping generation of WM_TIME events */
                PaintImage();                               /* redraw original image, erasing animation */
            }
            if ((TCHAR)wParam == '3')
            {
                ThreadRunning = 1;
                _beginthread(AnimationThread,0,MainWnd);    /* start up a child thread to do other work while this thread continues GUI */
            }
            if ((TCHAR)wParam == '4')
            {
                ThreadRunning = 0;
            }
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;

        case WM_TIMER:    /* this event gets triggered every time the timer goes off */
            hDC=GetDC(MainWnd);
            SetPixel(hDC,TimerCol,TimerRow,RGB(0,0,255));   /* color the animation pixel blue */
            ReleaseDC(MainWnd,hDC);
            TimerRow++;
            TimerCol+=2;
            break;

        case WM_HSCROLL:      /* this event could be used to change what part of the image to draw */
            PaintImage();     /* direct PaintImage calls eliminate flicker; the alternative is InvalidateRect(hWnd,NULL,TRUE); UpdateWindow(hWnd); */
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;

        case WM_VSCROLL:      /* this event could be used to change what part of the image to draw */
            PaintImage();
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return(DefWindowProc(hWnd,uMsg,wParam,lParam));
            break;
    }

    hMenu=GetMenu(MainWnd);

    /* CHECKS AND UNCHECKS IF SELECTED OPTION ON GUI FOR SHOWING PIXEL COORDINATES
        PLAY MODE, AND STEP MODE */
    if (ShowPixelCoords == 1)
    {
        CheckMenuItem(hMenu, ID_SHOWPIXELCOORDS, MF_CHECKED);   /* you can also call EnableMenuItem() to grey(disable) an option */
    }
    else
    {
        CheckMenuItem(hMenu, ID_SHOWPIXELCOORDS, MF_UNCHECKED);
    }
    if (play_mode == 1)
    {
        CheckMenuItem(hMenu, ID_PLAY_MODE, MF_CHECKED);
    }
    else
    {
        CheckMenuItem(hMenu, ID_PLAY_MODE,  MF_UNCHECKED);
    }
    if (step_mode == 1)
    {
        CheckMenuItem(hMenu, ID_STEP_MODE, MF_CHECKED);
    }
    else
    {
        CheckMenuItem(hMenu, ID_STEP_MODE, MF_UNCHECKED);
    }

    DrawMenuBar(hWnd);

    return(0L);
}


/* FUNCTION GOES BACK TO ORIGINAL IMAGE */
void PaintImage()

{
    PAINTSTRUCT         Painter;
    HDC                 hDC;
    BITMAPINFOHEADER    bm_info_header;
    BITMAPINFO          *bm_info;
    int                 i,r,c,DISPLAY_ROWS,DISPLAY_COLS;
    unsigned char       *DisplayImage;

    if (OriginalImage == NULL)
    {
        return;     /* no image to draw */
    }

    /* Windows pads to 4-byte boundaries.  We have to round the size up to 4 in each dimension, filling with black. */
    DISPLAY_ROWS = ROWS;
    DISPLAY_COLS = COLS;

    if (DISPLAY_ROWS % 4 != 0)
    {
        DISPLAY_ROWS = (DISPLAY_ROWS / 4 + 1) * 4;
    }
    if (DISPLAY_COLS % 4 != 0)
    {
        DISPLAY_COLS = (DISPLAY_COLS / 4 + 1) * 4;
    }

    DisplayImage = (unsigned char *)calloc(DISPLAY_ROWS*DISPLAY_COLS,1);

    for (r = 0; r < ROWS; r++)
    {
        for (c = 0; c < COLS; c++)
        {
            DisplayImage[r*DISPLAY_COLS + c] = OriginalImage[r*COLS + c];
        }
    }

    BeginPaint(MainWnd,&Painter);
    hDC=GetDC(MainWnd);
    bm_info_header.biSize=sizeof(BITMAPINFOHEADER);
    bm_info_header.biWidth=DISPLAY_COLS;
    bm_info_header.biHeight=-DISPLAY_ROWS;
    bm_info_header.biPlanes=1;
    bm_info_header.biBitCount=8;
    bm_info_header.biCompression=BI_RGB;
    bm_info_header.biSizeImage=0;
    bm_info_header.biXPelsPerMeter=0;
    bm_info_header.biYPelsPerMeter=0;
    bm_info_header.biClrUsed=256;
    bm_info_header.biClrImportant=256;
    bm_info=(BITMAPINFO *)calloc(1,sizeof(BITMAPINFO) + 256*sizeof(RGBQUAD));
    bm_info->bmiHeader=bm_info_header;

    for (i=0; i<256; i++)
    {
          bm_info->bmiColors[i].rgbBlue=bm_info->bmiColors[i].rgbGreen=bm_info->bmiColors[i].rgbRed=i;
          bm_info->bmiColors[i].rgbReserved=0;
    }

    SetDIBitsToDevice(hDC,0,0,DISPLAY_COLS,DISPLAY_ROWS,0,0,0, /* first scan line */DISPLAY_ROWS, /* number of scan lines */DisplayImage,bm_info,DIB_RGB_COLORS);
    ReleaseDC(MainWnd,hDC);
    EndPaint(MainWnd,&Painter);

    free(DisplayImage);
    free(bm_info);
}


```

```c
void AnimationThread(HWND AnimationWindowHandle)
{
    HDC     hDC;
    char    text[300];

    ThreadRow = ThreadCol = 0;
    while (ThreadRunning == 1)
    {
          hDC=GetDC(MainWnd);
          SetPixel(hDC,ThreadCol,ThreadRow,RGB(0,255,0));   /* color the animation
          pixel green */
          sprintf(text,"%d, %d",ThreadRow,ThreadCol);
          TextOut(hDC,300,0,text,strlen(text));     /* draw text on the window */
          ReleaseDC(MainWnd,hDC);
          ThreadRow+=3;
          ThreadCol++;
          Sleep(100);       /* pause 100 ms */
    }
}

/* REGION GROW FUNCTION */
void region_grow(HWND play_mode_window_handle)
{
    // Variable Declaration Section
    HDC hDC;
    int row, col;
    int x_pos = mouse_x_pos;
    int y_pos = mouse_y_pos;
    int has_been_painted = 0;
    int queue[MAX_QUEUE];
    int queue_head, queue_tail;
    int average;
    int total;
    int count;
    int index;
    int i;
    int x_1, y_1, x_2, y_2;
    int distance;
    int thread_num = total_threads;
    unsigned char *labels;

    labels = (unsigned char *)calloc(ROWS * COLS, sizeof(unsigned char));

    average = total = (int)OriginalImage[y_pos*COLS + x_pos];

    index = (y_pos * COLS) + x_pos;
    labels[index] = 1;
    queue[0] = index;

    queue_head = 1;
    queue_tail = 0;
    count = 1;
    is_j_pressed = 0;

    x_1 = x_pos; // Original X value for Centroid COLS
    y_1 = y_pos; // Original Y value for Centroid ROWS
    x_2 = x_pos;
    y_2 = y_pos;


    while (queue_head != queue_tail && fill_thread_running == 1)
    {
        hDC = GetDC(MainWnd);
        // Recalculate the average every 50 pixels
        if ((count % 50) == 0)
        {
            average = total / count;
        }
        /* STOPS ALL THREADS IF STEP MODE AND PLAY MODE ARE NOT SELECTED */
        while (step_mode == 0 && play_mode == 0)
        {
            if (end_thread == 1 && (total_threads == thread_num))
            {
                _endthread();
            }
        }
        for (row = -1; row <= 1; row++)
        {
            for (col = -1; col <= 1; col++)
            {
                if ((row == 0) && (col == 0))
                {
                    continue;
                }
                if ((queue[queue_tail] / COLS + row) < 0 ||
                    (queue[queue_tail] / COLS + row) >= ROWS ||
                    (queue[queue_tail] % COLS + col) < 0 ||
                    (queue[queue_tail] % COLS + col) >= COLS)
                {
                    continue;
                }
                if (labels[(queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                COLS + col] != has_been_painted)
                {
                    continue;
                }

                /* KILLS LAST CREATED THREAD*/
                if (end_thread == 1 && (total_threads == thread_num))
                {
                    _endthread();
                }

                /* TEST CRITERIA TO SEE IF PIXEL CAN BE JOINED */
                index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                COLS + col;
                if (abs((int)(OriginalImage[index] - average)) > thresh)
                {
                    continue;
                }

                /* CENTROID CRITERIA */
                x_2 = x_1 / count; // COL
                y_2 = y_1 / count; // ROW
                distance = sqrt(SQR((queue[queue_tail] / COLS + row) - y_2) +
                SQR((queue[queue_tail] % COLS + col) - x_2));
                if (distance > rad)
                {
                    continue;
                }
                x_1 += queue[queue_tail] % COLS + col;
                y_1 += queue[queue_tail] / COLS + row;

                /* PAINTS CURRENT IMAGE BASED ON SELECTED PIXELS AND SELECTED COLOR */
                SetPixel(hDC, queue[queue_tail] % COLS + col, queue[queue_tail] / COLS
                + row,
                    RGB(GetRValue(rgbCurrent), GetGValue(rgbCurrent),
                    GetBValue(rgbCurrent)));

                /* IF PIXEL HAS BEEN PAINTED, THREAD NUMBER WHICH PAINTED IT IS SAVED
                TO ARRAY FOR UNDO PURPOSES */
                if (indices[(queue[queue_tail] / COLS + row) * COLS +
                (queue[queue_tail] % COLS + col)] == 0)
                {
                    indices[(queue[queue_tail] / COLS + row) * COLS +
                    (queue[queue_tail] % COLS + col)] = thread_num;
                }

```

```
                    /* LABELS KEEPS TRACK IF PIXEL HAS BEEN PAINTED OR NOT IN ORDER TO SKIP
                    IF IT HAS*/
                    index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                    COLS + col;
                    labels[index] = 1;

                    total += OriginalImage[index];
                    count++;

                    index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
                    COLS + col;
                    queue[queue_head] = index;
                    queue_head = (queue_head + 1) % MAX_QUEUE;

                    if (queue_head == queue_tail)
                    {
                        exit(0);
                    }

                    /* DIFFERENCES BETWEEN PROCEDURE OF PLAY MODE AND STEP MODE */
                    if (play_mode == 1)
                    {
                        Sleep(1);
                    }
                    /* STUCK ON WHILE LOOP WHILE USER DOES NOT PRESS J ON STEP MODE */
                    else
                    {
                        while ((is_j_pressed == 0) && (step_mode == 1)) {}
                        Sleep(1);
                        is_j_pressed = 0;
                    }

                }
            }
            queue_tail = (queue_tail + 1) % MAX_QUEUE;
            ReleaseDC(MainWnd, hDC);
        }
        _endthread();
    }
```