

```

1  /*
2      Name: Rodrigo Ignacio Rojas Garcia
3      Lab#: 4
4  */
5
6  // Library Declaration Section
7  #include <stdio.h>
8  #include <math.h>
9  #include <time.h>
10 #include <sys/timeb.h>
11 #include <windows.h>
12 #include <wingdi.h>
13 #include <winuser.h>
14 #include <process.h>
15 #include "resource.h"
16 #include "globals.h"
17
18 int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine,
19 int nCmdShow)
20 {
21     MSG          msg;
22     HWND         hWnd;
23     WNDCLASS     wc;
24
25     wc.style=CS_HREDRAW | CS_VREDRAW;
26     wc.lpfnWndProc=(WNDPROC)WndProc;
27     wc.cbClsExtra=0;
28     wc.cbWndExtra=0;
29     wc.hInstance=hInstance;
30     wc.hIcon=LoadIcon(hInstance,"ID_PLUS_ICON");
31     wc.hCursor=LoadCursor(NULL,IDC_ARROW);
32     wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
33     wc.lpszMenuName="ID_MAIN_MENU";
34     wc.lpszClassName="PLUS";
35
36     if (!RegisterClass(&wc))
37     {
38         return(FALSE);
39     }
40
41     hWnd=CreateWindow("PLUS","plus program", WS_OVERLAPPEDWINDOW | WS_HSCROLL |
42     WS_VSCROLL, CW_USEDEFAULT,0,400,400,NULL,NULL,hInstance,NULL);
43     if (!hWnd)
44     {
45         return(FALSE);
46     }
47
48     ShowScrollBar(hWnd,SB_BOTH,FALSE);
49     ShowWindow(hWnd,nCmdShow);
50     UpdateWindow(hWnd);
51     MainWnd=hWnd;
52
53     /* SETS GLOBAL VARIABLES ONCE PROGRAM STARTS */
54     ShowPixelCoords=0;
55     play_mode = 0;
56     step_mode = 0;
57     total_threads = 0;
58
59     strcpy(filename,"");
60     OriginalImage=NULL;
61     ROWS=COLS=0;
62
63     InvalidateRect(hWnd,NULL,TRUE);
64     UpdateWindow(hWnd);
65
66     while (GetMessage(&msg,NULL,0,0))
67     {
68         TranslateMessage(&msg);
69         DispatchMessage(&msg);
70     }
71 }

```

```

68     }
69
70     return(msg.wParam);
71 }
72
73 /* CALLBACK FOR PIXEL INTENSITY*/
74 LRESULT CALLBACK WndProc2(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
75 {
76     switch (uMsg)
77     {
78     case WM_COMMAND:
79         switch (LOWORD(wParam))
80         {
81             case IDOK:
82                 GetDlgItemText(hWnd, IDC_EDIT1, threshold, 256);
83                 thresh = atoi(threshold);
84                 EndDialog(hWnd, wParam);
85                 break;
86             case IDCANCEL:
87                 EndDialog(hWnd, wParam);
88         }
89     }
90     return(0L);
91 }
92
93 /* CALLBACK FOR CENTROID DISTANCE */
94 LRESULT CALLBACK WndProc3(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
95 {
96     switch (uMsg)
97     {
98     case WM_COMMAND:
99         switch (LOWORD(wParam))
100         {
101             case IDOK:
102                 GetDlgItemText(hWnd, IDC_EDIT1, radius, 256);
103                 rad = atoi(radius);
104                 EndDialog(hWnd, wParam);
105                 break;
106             case IDCANCEL:
107                 EndDialog(hWnd, wParam);
108         }
109     }
110     return(0L);
111 }
112
113 LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
114 {
115     // Variable Declaration Section
116     HMENU          hMenu;
117     OPENFILENAME    ofn;
118     FILE            *fpt;
119     HDC             hDC;
120     char            header[320],text[320];
121     int             BYTES,xPos,yPos;
122
123     switch (uMsg)
124     {
125     case WM_COMMAND:
126         switch (LOWORD(wParam))
127         {
128             /* EVENT WHEN USER SELECT SHOW PIXEL */
129             case ID_SHOWPIXELCOORDS:
130                 ShowPixelCoords = (ShowPixelCoords + 1) % 2;
131                 //PaintImage();
132                 break;
133
134             /* EVENT WHEN USER SELECTS PLAY MODE*/
135             case ID_PLAY_MODE:
136                 play_mode = (play_mode + 1) % 2;

```

```

137         step_mode = 0;
138         end_thread = 0;
139         break;
140
141     /* EVENT WHEN USER SELECTS STEP MODE*/
142     case ID_STEP_MODE:
143         step_mode = (step_mode + 1) % 2;
144         play_mode = 0;
145         end_thread = 0;
146         break;
147
148     /* CREATES DIALOG BOX FOR COLORS*/
149     case ID_SELECT_COLOR:
150         ZeroMemory(&color, sizeof(CHOOSECOLOR));
151         color.lStructSize = sizeof(CHOOSECOLOR);
152         color.hwndOwner = hWnd;
153         color.lpCustColors = (LPDWORD)acrCustClr;
154         color.rgbResult = rgbCurrent;
155         color.Flags = CC_FULLOPEN | CC_RGBINIT;
156         if (ChooseColor(&color) == TRUE)
157         {
158             hbrush = CreateSolidBrush(color.rgbResult);
159             rgbCurrent = color.rgbResult;
160         }
161         break;
162
163     /* EVENT WHEN USER SELECTS UNDO */
164     case ID_UNDO:
165         end_thread = 1;
166         int i, j;
167         HDC hDC;
168         /* UNDO'S LAST THREAD DRAWING BY SETTING PIXELS CHANGED TO ORIGINAL
169         IMAGE RGB VALUES*/
170         for (i = 0; i < ROWS; i++)
171         {
172             for (j = 0; j < COLS; j++)
173             {
174                 if (indices[i * COLS + j] == total_threads)
175                 {
176                     hDC = GetDC(MainWnd);
177                     SetPixel(hDC, j, i, RGB(OriginalImage[i * COLS + j],
178                     OriginalImage[i * COLS + j], OriginalImage[i * COLS
179                     + j]));
180                     ReleaseDC(MainWnd, hDC);
181                     indices[i * COLS + j] = 0;
182                 }
183             }
184             end_thread = 0;
185             if (total_threads > 0)
186             {
187                 total_threads -= 1;
188             }
189             break;
190
191     /* EVENT WHEN USER SELECTS TO CHANGE PIXEL INTENSITY*/
192     case ID_PIXEL_INTENSITY:
193         DialogBox(NULL, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, WndProc2);
194         break;
195
196     /* EVENT WHEN USER SELECTS TO CHANGE THE CENTROID DISTANCE */
197     case ID_CENTROID_DISTANCE:
198         DialogBox(NULL, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, WndProc3);
199         break;
200
201     /* EVENT WHEN USER SELECTS TO LOAD A PICTURE TO THE GUI*/
202     case ID_FILE_LOAD:
203         if (OriginalImage != NULL)
204         {

```

```

204         free(OriginalImage);
205         OriginalImage = NULL;
206     }
207     memset(&(ofn), 0, sizeof(ofn));
208     ofn.lStructSize = sizeof(ofn);
209     ofn.lpstrFile = filename;
210     filename[0] = 0;
211     ofn.nMaxFile = MAX_FILENAME_CHARS;
212     ofn.Flags = OFN_EXPLORER | OFN_HIDEREADONLY;
213     ofn.lpstrFilter = "PPM files\0*.ppm\0All files\0*.*\0\0";
214     if (!(GetOpenFileName(&ofn)) || filename[0] == '\0')
215     {
216         break;          /* user cancelled load */
217     }
218     if ((fpt = fopen(filename, "rb")) == NULL)
219     {
220         MessageBox(NULL, "Unable to open file", filename, MB_OK |
221             MB_APPLMODAL);
222         break;
223     }
224     fscanf(fpt, "%s %d %d %d", header, &COLS, &ROWS, &BYTES);
225     if (strcmp(header, "P5") != 0 || BYTES != 255)
226     {
227         MessageBox(NULL, "Not a PPM (P5 greyscale) image", filename,
228             MB_OK | MB_APPLMODAL);
229         fclose(fpt);
230         break;
231     }
232     OriginalImage = (unsigned char *)calloc(ROWS*COLS, 1);
233     header[0] = fgetc(fpt); /* whitespace character after header */
234     fread(OriginalImage, 1, ROWS*COLS, fpt);
235     fclose(fpt);
236     SetWindowText(hWnd, filename);
237     PaintImage();
238
239     /* CREATES A GLOBAL VARIABLE FOR INDICES*/
240     indices = (int *)calloc(ROWS * COLS, sizeof(int));
241
242     break;
243
244     case ID_FILE_QUIT:
245         DestroyWindow(hWnd);
246         break;
247
248     case WM_SIZE:          /* could be used to detect when window size changes */
249         PaintImage();
250         return(DefWindowProc(hWnd, uMsg, wParam, lParam));
251         break;
252
253     case WM_PAINT:
254         PaintImage();
255         return(DefWindowProc(hWnd, uMsg, wParam, lParam));
256         break;
257
258     case WM_LBUTTONDOWN:case WM_RBUTTONDOWN:
259
260         /* MOUSE CLICK FOR PLAY MODE OR STEP MODE */
261         if ((play_mode == 1) || (step_mode == 1))
262         {
263             /* GETS THE X AND Y POSITION OF CLICKED POSITION IN IMAGE*/
264             mouse_x_pos = LOWORD(lParam);
265             mouse_y_pos = HIWORD(lParam);
266
267             /* CREATES A THREAD AND BEGINS REGION GROW ON IMAGE */
268             if ((mouse_x_pos >= 0) && (mouse_x_pos < COLS) && (mouse_y_pos >= 0) &&
269                 (mouse_y_pos < ROWS));
270             {

```

```

270         fill_thread_running = 1;
271         _beginthread(region_grow, 0, MainWnd);
272         total_threads += 1;
273     }
274
275 }
276 return(DefWindowProc(hWnd,uMsg,wParam,lParam));
277 break;
278
279 /* EVENT THAT SHOWS CURRENT CURSOR PIXEL AND DRAWS ON IMAGE */
280 case WM_MOUSEMOVE:
281     if (ShowPixelCoords == 1)
282     {
283         xPos=LOWORD(lParam);
284         yPos=HIWORD(lParam);
285         if (xPos >= 0 && xPos < COLS && yPos >= 0 && yPos < ROWS)
286         {
287             sprintf(text,"%d, %d => %d", xPos, yPos,
288                 OriginalImage[yPos*COLS+xPos]);
289             hDC=GetDC(MainWnd);
290             TextOut(hDC,0,0,text,strlen(text)); /* draw text on the
291                 window */
292             SetPixel(hDC,xPos,yPos,RGB(255,0,0)); /* color the cursor
293                 position red */
294             ReleaseDC(MainWnd,hDC);
295         }
296     }
297 return(DefWindowProc(hWnd,uMsg,wParam,lParam));
298 break;
299
300 case WM_KEYDOWN:
301     if (wParam == 's' || wParam == 'S')
302     {
303         PostMessage(MainWnd, WM_COMMAND, ID_SHOWPIXELCOORDS, 0); /* send
304             message to self */
305     }
306     /* VENT THAT TRIGGERS WHEN USER PRESSES J ON STEP MODE, THE REGION KEEPS
307     GROWING */
308     if (wParam == 'j' || wParam == 'J')
309     {
310         is_j_pressed = 1;
311     }
312     if ((TCHAR)wParam == '1')
313     {
314         TimerRow=TimerCol=0;
315         SetTimer(MainWnd,TIMER_SECOND,10,NULL); /* start up 10 ms timer */
316     }
317     if ((TCHAR)wParam == '2')
318     {
319         KillTimer(MainWnd,TIMER_SECOND); /* halt timer, stopping
320             generation of WM_TIME events */
321         PaintImage(); /* redraw original image,
322             erasing animation */
323     }
324     if ((TCHAR)wParam == '3')
325     {
326         ThreadRunning = 1;
327         _beginthread(AnimationThread,0,MainWnd); /* start up a child thread
328             to do other work while this thread continues GUI */
329     }
330     if ((TCHAR)wParam == '4')
331     {
332         ThreadRunning = 0;
333     }
334 return(DefWindowProc(hWnd,uMsg,wParam,lParam));
335 break;
336
337 case WM_TIMER: /* this event gets triggered every time the timer goes off */
338     hDC=GetDC(MainWnd);

```

```

331         SetPixel(hDC,TimerCol,TimerRow,RGB(0,0,255));    /* color the animation
pixel blue */
332         ReleaseDC(MainWnd,hDC);
333         TimerRow++;
334         TimerCol+=2;
335         break;
336
337     case WM_HSCROLL:          /* this event could be used to change what part of the
image to draw */
338         PaintImage();        /* direct PaintImage calls eliminate flicker; the
alternative is InvalidateRect(hWnd,NULL,TRUE); UpdateWindow(hWnd); */
339         return(DefWindowProc(hWnd,uMsg,wParam,lParam));
340         break;
341
342     case WM_VSCROLL:          /* this event could be used to change what part of the
image to draw */
343         PaintImage();
344         return(DefWindowProc(hWnd,uMsg,wParam,lParam));
345         break;
346
347     case WM_DESTROY:
348         PostQuitMessage(0);
349         break;
350     default:
351         return(DefWindowProc(hWnd,uMsg,wParam,lParam));
352         break;
353 }
354
355 hMenu=GetMenu(MainWnd);
356
357 /* CHECKS AND UNCHECKS IF SELECTED OPTION ON GUI FOR SHOWING PIXEL COORDINATES
PLAY MODE, AND STEP MODE */
358
359 if (ShowPixelCoords == 1)
360 {
361     CheckMenuItem(hMenu, ID_SHOWPIXELCOORDS, MF_CHECKED);    /* you can also call
EnableMenuItem() to grey(disable) an option */
362 }
363 else
364 {
365     CheckMenuItem(hMenu, ID_SHOWPIXELCOORDS, MF_UNCHECKED);
366 }
367 if (play_mode == 1)
368 {
369     CheckMenuItem(hMenu, ID_PLAY_MODE, MF_CHECKED);
370 }
371 else
372 {
373     CheckMenuItem(hMenu, ID_PLAY_MODE, MF_UNCHECKED);
374 }
375 if (step_mode == 1)
376 {
377     CheckMenuItem(hMenu, ID_STEP_MODE, MF_CHECKED);
378 }
379 else
380 {
381     CheckMenuItem(hMenu, ID_STEP_MODE, MF_UNCHECKED);
382 }
383
384 DrawMenuBar(hWnd);
385
386 return(0L);
387 }
388
389
390 /* FUNCTION GOES BACK TO ORIGINAL IMAGE */
391 void PaintImage()
392 {
393     PAINTSTRUCT        Painter;
394

```

```

395     HDC                hDC;
396     BITMAPINFOHEADER    bm_info_header;
397     BITMAPINFO           *bm_info;
398     int                 i,r,c,DISPLAY_ROWS,DISPLAY_COLS;
399     unsigned char        *DisplayImage;
400
401     if (OriginalImage == NULL)
402     {
403         return;        /* no image to draw */
404     }
405
406     /* Windows pads to 4-byte boundaries.  We have to round the size up to 4 in each
407     dimension, filling with black. */
408     DISPLAY_ROWS = ROWS;
409     DISPLAY_COLS = COLS;
410
411     if (DISPLAY_ROWS % 4 != 0)
412     {
413         DISPLAY_ROWS = (DISPLAY_ROWS / 4 + 1) * 4;
414     }
415     if (DISPLAY_COLS % 4 != 0)
416     {
417         DISPLAY_COLS = (DISPLAY_COLS / 4 + 1) * 4;
418     }
419
420     DisplayImage = (unsigned char *)calloc(DISPLAY_ROWS*DISPLAY_COLS,1);
421
422     for (r = 0; r < ROWS; r++)
423     {
424         for (c = 0; c < COLS; c++)
425         {
426             DisplayImage[r*DISPLAY_COLS + c] = OriginalImage[r*COLS + c];
427         }
428     }
429
430     BeginPaint(MainWnd,&Painter);
431     hDC=GetDC(MainWnd);
432     bm_info_header.biSize=sizeof(BITMAPINFOHEADER);
433     bm_info_header.biWidth=DISPLAY_COLS;
434     bm_info_header.biHeight=-DISPLAY_ROWS;
435     bm_info_header.biPlanes=1;
436     bm_info_header.biBitCount=8;
437     bm_info_header.biCompression=BI_RGB;
438     bm_info_header.biSizeImage=0;
439     bm_info_header.biXPelsPerMeter=0;
440     bm_info_header.biYPelsPerMeter=0;
441     bm_info_header.biClrUsed=256;
442     bm_info_header.biClrImportant=256;
443     bm_info=(BITMAPINFO *)calloc(1,sizeof(BITMAPINFO) + 256*sizeof(RGBQUAD));
444     bm_info->bmiHeader=bm_info_header;
445
446     for (i=0; i<256; i++)
447     {
448         bm_info->bmiColors[i].rgbBlue=bm_info->bmiColors[i].rgbGreen=bm_info->bmiColors
449         [i].rgbRed=i;
450         bm_info->bmiColors[i].rgbReserved=0;
451     }
452
453     SetDIBitsToDevice(hDC,0,0,DISPLAY_COLS,DISPLAY_ROWS,0,0,0, /* first scan line
454     *//DISPLAY_ROWS, /* number of scan lines *//DisplayImage,bm_info,DIB_RGB_COLORS);
455     ReleaseDC(MainWnd,hDC);
456     EndPaint(MainWnd,&Painter);
457
458     free(DisplayImage);
459     free(bm_info);
460 }

```

```

460 void AnimationThread(HWND AnimationWindowHandle)
461 {
462     HDC      hDC;
463     char      text[300];
464
465     ThreadRow = ThreadCol = 0;
466     while (ThreadRunning == 1)
467     {
468         hDC=GetDC(MainWnd);
469         SetPixel(hDC,ThreadCol,ThreadRow,RGB(0,255,0));    /* color the animation
pixel green */
470         sprintf(text,"%d, %d",ThreadRow,ThreadCol);
471         TextOut(hDC,300,0,text,strlen(text));    /* draw text on the window */
472         ReleaseDC(MainWnd,hDC);
473         ThreadRow+=3;
474         ThreadCol++;
475         Sleep(100);    /* pause 100 ms */
476     }
477 }
478
479 /* REGION GROW FUNCTION */
480 void region_grow(HWND play_mode_window_handle)
481 {
482     // Variable Declaration Section
483     HDC hDC;
484     int row, col;
485     int x_pos = mouse_x_pos;
486     int y_pos = mouse_y_pos;
487     int has_been_painted = 0;
488     int queue[MAX_QUEUE];
489     int queue_head, queue_tail;
490     int average;
491     int total;
492     int count;
493     int index;
494     int i;
495     int x_1, y_1, x_2, y_2;
496     int distance;
497     int thread_num = total_threads;
498     unsigned char *labels;
499
500     labels = (unsigned char *)calloc(ROWS * COLS, sizeof(unsigned char));
501
502     average = total = (int)OriginalImage[y_pos*COLS + x_pos];
503
504     index = (y_pos * COLS) + x_pos;
505     labels[index] = 1;
506     queue[0] = index;
507
508     queue_head = 1;
509     queue_tail = 0;
510     count = 1;
511     is_j_pressed = 0;
512
513     x_1 = x_pos; // Original X value for Centroid COLS
514     y_1 = y_pos; // Original Y value for Centroid ROWS
515     x_2 = x_pos;
516     y_2 = y_pos;
517
518
519     while (queue_head != queue_tail && fill_thread_running == 1)
520     {
521         hDC = GetDC(MainWnd);
522         // Recalculate the average every 50 pixels
523         if ((count % 50) == 0)
524         {
525             average = total / count;
526         }
527         /* STOPS ALL THREADS IF STEP MODE AND PLAY MODE ARE NOT SELECTED */

```



```

528 while (step_mode == 0 && play_mode == 0)
529 {
530     if (end_thread == 1 && (total_threads == thread_num))
531     {
532         _endthread();
533     }
534 }
535 for (row = -1; row <= 1; row++)
536 {
537     for (col = -1; col <= 1; col++)
538     {
539         if ((row == 0) && (col == 0))
540         {
541             continue;
542         }
543         if ((queue[queue_tail] / COLS + row) < 0 ||
544             (queue[queue_tail] / COLS + row) >= ROWS ||
545             (queue[queue_tail] % COLS + col) < 0 ||
546             (queue[queue_tail] % COLS + col) >= COLS)
547         {
548             continue;
549         }
550         if (labels[(queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
551             COLS + col] != has_been_painted)
552         {
553             continue;
554         }
555         /* KILLS LAST CREATED THREAD*/
556         if (end_thread == 1 && (total_threads == thread_num))
557         {
558             _endthread();
559         }
560
561         /* TEST CRITERIA TO SEE IF PIXEL CAN BE JOINED */
562         index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
563             COLS + col;
564         if (abs((int)(OriginalImage[index] - average)) > thresh)
565         {
566             continue;
567         }
568
569         /* CENTROID CRITERIA */
570         x_2 = x_1 / count; // COL
571         y_2 = y_1 / count; // ROW
572         distance = sqrt(SQR((queue[queue_tail] / COLS + row) - y_2) +
573             SQR((queue[queue_tail] % COLS + col) - x_2));
574         if (distance > rad)
575         {
576             continue;
577         }
578         x_1 += queue[queue_tail] % COLS + col;
579         y_1 += queue[queue_tail] / COLS + row;
580
581         /* PAINTS CURRENT IMAGE BASED ON SELECTED PIXELS AND SELECTED COLOR */
582         SetPixel(hDC, queue[queue_tail] % COLS + col, queue[queue_tail] / COLS
583             + row,
584             RGB(GetRValue(rgbCurrent), GetGValue(rgbCurrent),
585                 GetBValue(rgbCurrent)));
586
587         /* IF PIXEL HAS BEEN PAINTED, THREAD NUMBER WHICH PAINTED IT IS SAVED
588            TO ARRAY FOR UNDO PURPOSES */
589         if (indices[(queue[queue_tail] / COLS + row) * COLS +
590             (queue[queue_tail] % COLS + col)] == 0)
591         {
592             indices[(queue[queue_tail] / COLS + row) * COLS +
593                 (queue[queue_tail] % COLS + col)] = thread_num;
594         }
595     }
596 }

```

```

589         /* LABELS KEEPS TRACK IF PIXEL HAS BEEN PAINTED OR NOT IN ORDER TO SKIP
590         IF IT HAS*/
591         index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
592         COLS + col;
593         labels[index] = 1;
594
595         total += OriginalImage[index];
596         count++;
597
598         index = (queue[queue_tail] / COLS + row)*COLS + queue[queue_tail] %
599         COLS + col;
600         queue[queue_head] = index;
601         queue_head = (queue_head + 1) % MAX_QUEUE;
602
603         if (queue_head == queue_tail)
604         {
605             exit(0);
606         }
607
608         /* DIFFERENCES BETWEEN PROCEDURE OF PLAY MODE AND STEP MODE */
609         if (play_mode == 1)
610         {
611             Sleep(1);
612         }
613         /* STUCK ON WHILE LOOP WHILE USER DOES NOT PRESS J ON STEP MODE */
614         else
615         {
616             while ((is_j_pressed == 0) && (step_mode == 1)) {}
617             Sleep(1);
618             is_j_pressed = 0;
619         }
620     }
621     queue_tail = (queue_tail + 1) % MAX_QUEUE;
622     ReleaseDC(MainWnd, hDC);
623 }
624 _endthread();
625 }
626

```