```c
/* Filename: letters_lab3.c
** Author: Netra Inamdar (C19486906)
** Date: 10/03/2019 */

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdbool.h>
#include<time.h>

int main(int argc, char* argv[])
{
    FILE            *fpt_1,*fpt_2,*fpt_image,*fpt_msf,*gt, *msf_original_image;
    unsigned char   *ori_image, *img, *marked_img, *img_copy, *eb_image;
    unsigned char   *msf_image, *final_msf, *MSF_copy;
    float           *mean_centered_template,*MSF;
    float           msf_min,msf_max,msf_sum;
    char            header[320],header1[320],header2[320],ch[1262];
    int             col_arr[1262],row_arr[1262];
    int             ROWS,COLS,BYTES,transitions,marked_arr[187];
    int             ROWS1,COLS1,BYTES1,edge_neighbors;
    int             ROWS2,COLS2,BYTES2,ROWS3,COLS3;
    int             r,c,r1,c1,r2,c2,sum,template_sum,mean_val;
    int             i,j,thresh,detected_count,not_detected_count;
    int             e_count=0,fp,tp,fn,tn,endpt,branchpt;
    float           TPR,FPR,tpr_array[256],fpr_array[256];
    int             edge_check,marked_arr_sum,iter_count;

    i=0;
    e_count=0;
    // Read and check original image:
    if ((fpt_image=fopen("parenthood.ppm","rb"))==NULL)
    {
        printf("Unable to open parenthood.ppm for reading.\n");
        exit(0);
    }
    fscanf(fpt_image,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);

    if(strcmp(header,"P5")!=0 || BYTES!=255)
    {
        printf("Not a greyscale 8-bit PPM image.\n");
        exit(0);
    }
    ori_image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    header[0]=fgetc(fpt_image); //read whitespace char that separates header
    fread(ori_image,1,COLS*ROWS,fpt_image);
    fclose(fpt_image);

    // Read and check original MSF image:
    if ((fpt_msf=fopen("MSF_original1.ppm","rb"))==NULL)
    {
        printf("Unable to open MSF_original1.ppm for reading.\n");
            exit(0);
    }
    fscanf(fpt_msf,"%s %d %d %d",header2,&COLS2,&ROWS2,&BYTES2);

    if(strcmp(header2,"P5")!=0 || BYTES2!=255)
    {
        printf("Not a greyscale 8-bit PPM image.\n");
        exit(0);
    }
    msf_image=(unsigned char *)calloc(ROWS2*COLS2,sizeof(unsigned char));
    header2[0]=fgetc(fpt_msf); //read whitespace char that separates header
    fread(msf_image,1,ROWS2*COLS2,fpt_msf);
    fclose(fpt_msf);

    // Read groundTruth file
    gt=fopen("parenthood_gt.txt","r");
    while(!feof(gt))
```

```
70          {
71                  fscanf(gt,"%s %d %d",&ch[i],&col_arr[i],&row_arr[i]);
72                  //printf("%s %d %d\n",ch,*col_arr,*row_arr);
73                  i++;
74          }
75          fclose(gt);
76
77      // Threshold Loop:
78      //thresh=255;
79      //while(thresh >=200)
80      for(thresh=255;thresh>=0;thresh--)
81      {
82          printf("Threshold:%d\t",thresh);
83          MSF_copy=(unsigned char *)calloc(ROWS2*COLS2,sizeof(unsigned char));
84          detected_count=0;
85          not_detected_count=0;
86          fp=0,tp=0,fn=0,tn=0;
87
88          // Create binary image based on threshold:
89          for(r=0;r<ROWS2;r++)
90          {
91              for(c=0;c<COLS2;c++)
92              {
93                  if(msf_image[r*COLS2+c]>thresh)
94                  {   MSF_copy[r*COLS2+c]=255; }
95                  else
96                  {   MSF_copy[r*COLS2+c]=0; }
97              }
98          }
99          //printf("after checking binary msf values\n");
100         e_count=0;
101         iter_count=0;
102         for(r=0;r<1262;r++)
103         {
104             COLS3=11;
105             ROWS3=17;
106             img=(unsigned char *)calloc(ROWS3*COLS3,sizeof(unsigned char));
107             img_copy=(unsigned char *)calloc(ROWS3*COLS3,sizeof(unsigned char));
108             marked_img=(unsigned char *)calloc(ROWS3*COLS3,sizeof(unsigned char));
109             eb_image=(unsigned char *)calloc(ROWS3*COLS3,sizeof(unsigned char));
110             //if(ch[r]=='e')
111             //{ e_count++;}
112             sum=0;
113             for(r2=-7;r2<=7;r2++)
114                 for(c2=-4;c2<=4;c2++)
115                     sum+= MSF_copy[(row_arr[r]+r2)*COLS2+ col_arr[r]+c2];
116
117             //iter_count=0;
118             if(sum<255)
119             {
120                 //printf("not detected!\n");
121                 not_detected_count++;
122                 if(ch[r]=='e')
123                 { ++fn; } // prior: tp
124                 else
125                 { ++tn; } // prior: fp
126                 continue;
127             }
128             //iter_count=0;
129             else if (sum>=255) // sum >=255 and hence detected
130             {
131                 //iter_count=0;
132
133
134                 //printf("detected!\n");
135                 //detected_count++;
136                 for(r1=1,r2=-7;r1<ROWS3-1,r2<=7;r1++,r2++)
137                 {
138                     for(c1=1,c2=-4;c1<COLS3-1,c2<=4;c1++,c2++)
```

```
                    {
                        img[r1*COLS3+c1]
                        = ori_image[(row_arr[r]+r2)*COLS+ col_arr[r]+c2];
                    }
                }

                // Threshold ori image copy at 128 to make binary image:
                for(r1=1;r1<ROWS3-1;r1++)
                        {
                                for(c1=1;c1<COLS3-1;c1++)
                                {
                                        if(img[r1*COLS3+c1]>=128)
                                        {       img[r1*COLS3+c1]=255;
                                                img_copy[r1*COLS3+c1]=255;

                                        }
                                        else
                                        {   img[r1*COLS3+c1]=0;
                                            img_copy[r1*COLS3+c1]=0;

                                        }
                                }
                        }
                /*
                // Save binary image:
                fpt_1=fopen("ori_copy_binary1.ppm","wb");
                fprintf(fpt_1,"P5 %d %d 255\n",COLS3,ROWS3);
                fwrite(img,COLS3*ROWS3,1,fpt_1);
                fclose(fpt_1);
                */

                for(r1=0;r1<ROWS3;r1++)
                {
                    img[r1*COLS3+0]=255;
                    img[r1*COLS3+10]=255;
                }
                for(c1=0;c1<COLS3;c1++)
                {
                    img[0*COLS3+c1]=255;
                    img[16*COLS3+c1]=255;
                }
                // Thin thresholded image to single pixel wide components:
                //iter_count+=1;
                while(true)
                {

                endpt=0;
                branchpt=0;
                for (i=0;i<187;i++)
                {
                    marked_arr[i]=0;
                }
                for(r1=1;r1<ROWS3-1;r1++)
                {
                 for(c1=1;c1<COLS3-1;c1++)
                 {
                    //endpt=0;
                    //branchpt=0;
                    if (img[r1*COLS3+c1]==0)
                    {
                    transitions=0;
                    if ((img[(r1-1)*COLS3+(c1-1)]-img[(r1-1)*COLS3+(c1)])==-255)
                        transitions+=1;
                    if
                    ((img[(r1-1)*COLS3+(c1)]-img[(r1-1)*COLS3+(c1+1)])==-255)

                       transitions+=1;
                    if
                    ((img[(r1-1)*COLS3+(c1+1)]-img[(r1)*COLS3+(c1+1)])==-255)
```

```
                    transitions+=1;
204                 if
                    ((img[(r1)*COLS3+(c1+1)]-img[(r1+1)*COLS3+(c1+1)])==-255)

                    transitions+=1;
205                 if
                    ((img[(r1+1)*COLS3+(c1+1)]-img[(r1+1)*COLS3+(c1)])==-255)

                    transitions+=1;
206                 if
                    ((img[(r1+1)*COLS3+(c1)]-img[(r1+1)*COLS3+(c1-1)])==-255)

                    transitions+=1;
207                 if
                    ((img[(r1+1)*COLS3+(c1-1)]-img[(r1)*COLS3+(c1-1)])==-255)

                    transitions+=1;
208                 if
                    ((img[(r1)*COLS3+(c1-1)]-img[(r1-1)*COLS3+(c1-1)])==-255)

                    transitions+=1;
209
210                 //printf("letter:%c\t",ch[r]);
211                 //printf("transitions:%d\t",transitions);
212                 if (transitions==1)  endpt+=1;
213                 //printf("endpt:%d\t",endpt);
214                 if (transitions>2) branchpt+=1;
215                 //printf("branchpt:%d\n",branchpt);
216
217                 //printf("%d\t%d\n",endpt,branchpt);
218
219                 edge_neighbors=-1;
220                 for(r2=-1;r2<=1;r2++)
221                  for(c2=-1;c2<=1;c2++)
222                  {
223                      if (img[(r1+r2)*COLS3+(c1+c2)]==0)
224                          edge_neighbors+=1;
225                  }
226                 //printf("r1:%d, c1:%d, transitions:%d\n",r1,c1,transitions);
227                 //if (img[r1*COLS3+c1]==0)
228                 //  edge_neighbors-=1;
229
230                 edge_check=0;
231                 if ( ((img[(r1-1)*COLS3+c1]!=0) || (img[(r1)*COLS3+(c1+1)]!=0)
232                    || ((img[(r1)*COLS3+(c1-1)]!=0) && (img[(r1+1)*COLS3+c1]!=0))))
233                     edge_check=1;
234
235                 if ( ((transitions==1) && (3<=edge_neighbors && edge_neighbors<=7)
                    && (edge_check==1))==true )
236                 {
237                     marked_img[r1*COLS3+c1]=1;
238                     //printf("marked r,c and ans:%d %d
                    %d\n",r1,c1,((transitions==1) && (3<=edge_neighbors &
                    edge_neighbors<=7) && (edge_check==1)));
239                     marked_arr[r1*COLS3+c1]=1;
240                 }
241
242                 } // if (edge is a pixel)
243
244                 eb_image[r1*COLS3+c1]=img[r1*COLS3+c1];
245                 if (endpt==1)
246                 {
247                     if(eb_image[r1*COLS3+c1]!=128)
248                         eb_image[r1*COLS3+c1]=128;
249                 }
250                 if (branchpt==1)
251                 {
252                     if(eb_image[r1*COLS3+c1]!=128)
```

```
                            eb_image[r1*COLS3+c1]=128;
                    }
                 }
                }

                for(r1=0;r1<ROWS3;r1++)
                {
                    for(c1=0;c1<COLS3;c1++)
                    {
                        if (marked_arr[r1*COLS3+c1]==1)
                        {
                            img[r1*COLS3+c1]=255;
                            //eb_image[r1*COLS3+c1]=255;
                        }
                    }
                }
                marked_arr_sum=0;
                for (i=0;i<187;i++)
                {
                    marked_arr_sum+=marked_arr[i];
                }

                if (marked_arr_sum==0) break;
                } //while(marked_arr_sum!=0);


                //printf("iter:%d,index:%d,endpt:%d,branchpt:%d,letter:%c\n",iter_count,r
                ,endpt,branchpt,ch[r]);


                if (endpt==1 && branchpt==1)
                {
                    //printf("detected letter is e and actual=%c.\n",ch[r]);
                    detected_count++;
                    if (ch[r]=='e') { ++tp; }
                    else if (ch[r]!='e') { ++fp; }
                }
                else
                {
                    not_detected_count++;
                    if (ch[r]=='e') { ++fn; }
                    else if (ch[r]!='e') { ++tn;
                    }
                }
            }
            ++iter_count;
            /*
            //save thinned image:
            fpt_1=fopen("thinned_image1.ppm","wb");
            fprintf(fpt_1,"P5 %d %d 255\n",COLS3,ROWS3);
            fwrite(img,COLS3*ROWS3,1,fpt_1);
            fclose(fpt_1);

            // save eb image:
            fpt_1=fopen("eb_image1.ppm","wb");
            fprintf(fpt_1,"P5 %d %d 255\n",COLS3,ROWS3);
            fwrite(eb_image,COLS3*ROWS3,1,fpt_1);
            fclose(fpt_1);
            */
        }

        ///*
        printf("detected:%d\t",detected_count);
        printf("not detected:%d\t",not_detected_count);
        printf("tp count:%d\t",tp);
        printf("fp count:%d\t",fp);
        printf("fn count:%d\t",fn);
        printf("tn count:%d\n",tn);
        //*/
```

```
319             TPR=((float)(tp)) / ((float)(tp+fn));
320             FPR=((float)(fp)) / ((float)(fp+tn));
321             tpr_array[thresh]=TPR;
322             fpr_array[thresh]=FPR;
323             printf("TPR:%.7f\t",TPR);
324             printf("FPR:%.7f\n",FPR);
325         }
326         //printf("True e count:%d\t",e_count); // 151
327         //printf("True Not e count:%d\n",1262-e_count); //1111
328 
329         /*
330         printf("TPR VALUES:\n");
331         for(j=255;j>=0;j--)
332         {
333             printf("%.2f,\t",tpr_array[j]);
334         }
335         printf("\nFPR VALUES:\n");
336             for(j=255;j>=0;j--)
337             {
338                     printf("%.2f,\t",fpr_array[j]);
339             }
340         printf("\n");
341         */
342     }
343 
```