```c
/* Filename: optical_char_recognition.c
** Author: Netra Inamdar (C19486906)
** Date: 9/19/2019
** This program reads parenthood.ppm, and parenthood_e_template.ppm images.
** It creates a normalized Matched Spatial Filtered Image to detect a letter 'e'
** by cross correlation technique.
** The program also reports count of TP, FP, TN, FN for each threshold. */

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

int main(int argc, char* argv[])
{
    FILE            *fpt_1,*fpt_image,*fpt_template,*gt, *msf_original_image;
    unsigned char   *ori_image;
    unsigned char   *template, *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,denom;
    int             ROWS1,COLS1,BYTES1,ROWS2,COLS2,BYTES2;
    int             r,c,r2,c2,sum,e_count=0,fp,tp,fn,tn;
    int             i=0,thresh,detected_count,not_detected_count;
    float           TPR,FPR,tpr_array[256],fpr_array[256];

    // 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 template image:
    if ((fpt_template=fopen("parenthood_e_template.ppm","rb"))==NULL)
    {
        printf("Unable to open parenthood_e_template.ppm for reading.\n");
        exit(0);
    }
    fscanf(fpt_template,"%s %d %d %d",header1,&COLS1,&ROWS1,&BYTES1);

    if(strcmp(header1,"P5")!=0 || BYTES1!=255)
    {
        printf("Not a greyscale 8-bit PPM image.\n");
        exit(0);
    }
    template=(unsigned char *)calloc(ROWS1*COLS1,sizeof(unsigned char));
    header1[0]=fgetc(fpt_template); //read whitespace char that separates header
    fread(template,1,COLS1*ROWS1,fpt_template);
    fclose(fpt_template);

    mean_centered_template=(float *)calloc(ROWS1*COLS1,sizeof(float));
    final_msf=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

    //printf("rows1 and cols1:%d %d\n",ROWS1,COLS1); //rows:15 and colums:9
    //printf("mean centered template:\n");
```

```
70      msf_sum=0;
71      for(r=0;r<ROWS1;r++)
72      {
73          for(c=0;c<COLS1;c++)
74          {
75              msf_sum+=template[r*COLS1+c]; //sum of all values in template
76          }
77      }
78      //printf("msf mean:%f\n",msf_sum/135); // mean value is: 165.39
79      for(r=0;r<ROWS1;r++)
80      {
81          for(c=0;c<COLS1;c++)
82          {
83              mean_centered_template[r*COLS1+c]=template[r*COLS1+c]-165.39;
84              // creating mean centered template by subtracting mean from each pixel.
85          }
86      }
87
88      //allocate memory for MSF:
89      MSF=(float *)calloc(ROWS*COLS,sizeof(float));
90
91      // Build MSF image, skipping the border points:
92      for (r=7;r<ROWS-7;r++)
93      {
94          for (c=4;c<COLS-4;c++)
95          {
96              msf_sum=0;
97              for (r2=-7;r2<=7;r2++)
98                  for(c2=-4;c2<=4;c2++)
99                      msf_sum+= (ori_image[(r+r2)*COLS+(c+c2)])*
100                                 (mean_centered_template[(r2+7)*COLS1+(c2+4)]);
101             MSF[r*COLS+c]=msf_sum; // create basic MSF image using cross correlation
102         }
103     }
104     msf_min=MSF[4547]; // first value after skipping border points
105     msf_max=MSF[4547];
106
107     for(r=7;r<ROWS-7;r++)
108     {
109         for(c=4;c<COLS-4;c++)
110         {
111             if(MSF[r*COLS+c]>msf_max)
112                 msf_max=MSF[r*COLS+c]; // update max pixel value of template
113             else if(MSF[r*COLS+c]<msf_min)
114                 msf_min=MSF[r*COLS+c]; // update min pixel value of template
115         }
116     }
117     // Re-scaling to 0-255 range of values:
118     for(r=7;r<ROWS-7;r++)
119     {
120         for(c=4;c<COLS-4;c++)
121         {
122             final_msf[r*COLS+c]=(int)(((MSF[r*COLS+c]-msf_min)/(msf_max-msf_min))*255);
123             // create a normalized MSF image
124         }
125     }
126     // write out normalized MSF image to see result:
127     fpt_1=fopen("MSF_original1.ppm","wb");
128     fprintf(fpt_1,"P5 %d %d 255\n",COLS,ROWS);
129     fwrite(final_msf,COLS*ROWS,1,fpt_1);
130     fclose(fpt_1);
131
132     gt=fopen("parenthood_gt.txt","r"); // read groundtruth and store coordinates
133     while(!feof(gt))
134     {
135         fscanf(gt,"%s %d %d",&ch[i],&col_arr[i],&row_arr[i]);
136         i++;
137     }
138     fclose(gt);
```

```
139
140         // Threshold Loop:
141         thresh=255;          // thresholding from 255 to 0
142         while(thresh >=0)
143         {
144             // Read and check original MSF image:
145             if ((msf_original_image=fopen("MSF_original1.ppm","rb"))==NULL)
146             {
147                 printf("Unable to open MSF_original1.ppm for reading.\n");
148                 exit(0);
149             }
150             fscanf(msf_original_image,"%s %d %d %d",header2,&COLS2,&ROWS2,&BYTES2);
151
152             if(strcmp(header2,"P5")!=0 || BYTES2!=255)
153             {
154                 printf("Not a greyscale 8-bit PPM image.\n");
155                 exit(0);
156             }
157             MSF_copy=(unsigned char *)calloc(ROWS2*COLS2,sizeof(unsigned char));
158             header2[0]=fgetc(msf_original_image); //read whitespace char that separates
                header
159             fread(MSF_copy,1,ROWS2*COLS2,msf_original_image);
160             fclose(msf_original_image);
161
162             printf("Threshold:%d\t",thresh);
163             detected_count=0;
164             not_detected_count=0;
165             fp=0,tp=0,fn=0,tn=0;
166
167             for(r=0;r<ROWS2;r++)
168             {
169                 for(c=0;c<COLS2;c++)
170                 {
171                     if(MSF_copy[r*COLS2+c]>thresh)
172                     {   MSF_copy[r*COLS2+c]=255; } // binary MSF image based on threshold
173                     else
174                     {   MSF_copy[r*COLS2+c]=0; } // assign 0 if less than threshold
175                 }
176             }
177             /* Save binary image for required threshold, after analysing output:
178             fpt_1=fopen("MSF_binary1.ppm","wb");
179             fprintf(fpt_1,"P5 %d %d 255\n",COLS2,ROWS2);
180             fwrite(MSF_copy,COLS2*ROWS2,1,fpt_1);
181             fclose(fpt_1);
182             */
183
184             e_count=0;
185             for(r=0;r<1262;r++)
186             {
187                 if(ch[r]=='e')
188                 { e_count++;}  // count actual number of letter 'e' from groundtruth
189                 sum=0;
190                 for(r2=-7;r2<=7;r2++)
191                     for(c2=-4;c2<=4;c2++)
192                         sum+= MSF_copy[(row_arr[r]+r2)*COLS2+ col_arr[r]+c2];
193
194                 if(sum>=255) // check if sum is greater than 255 to detect letter e
195                 {
196                     //printf("detected!\n");
197                     detected_count++;
198                     if(ch[r]=='e') //check if letter is really 'e' from groundtruth
199                         { ++tp; } // update count of True positives
200                     else
201                         { ++fp; } // update count of false positives
202                 }
203                 else
204                 {
205                     //printf("not detected!\n");
206                     not_detected_count++;
```

```
                    if(ch[r]=='e')
                        { ++fn; } // update count of false negatives
                    else
                        { ++tn; } // update count of true negatives
                }
            }
            //printf("detected:%d\t",detected_count);
            //printf("not detected:%d\t",not_detected_count);
            printf("tp count:%d\t",tp);
            printf("fp count:%d\n",fp);
            //printf("fn count:%d\t",fn);
            //printf("tn count:%d\n",tn);

            TPR=((float)(tp)) / ((float)(tp+fn)); // calculate True positive rate
            FPR=((float)(fp)) / ((float)(fp+tn)); // calculate false positive rate
            tpr_array[thresh]=TPR;
            fpr_array[thresh]=FPR;
            //printf("TPR:%.7f\t",TPR);
            //printf("FPR:%.7f\n",FPR);

            thresh--; // decrement threshold value
        }
        //printf("True e count:%d\t",e_count); // 151
        //printf("True Not e count:%d\n",1262-e_count); //1111

        //printf("TPR VALUES:\n");
        for(i=255;i>=0;i--)
        {
            //printf("%.2f,\t",tpr_array[i]); // To print all TPR values for ROC curve
        }
        //printf("\nFPR VALUES:\n");
        for(i=255;i>=0;i--)
        {
            //printf("%.2f,\t",fpr_array[i]); // To print all FPR values
        }
    }
```