PCB Quality Control: Code Challenge

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <unistd.h>
#include "bitmap.h"
#include <string.h>

#define BITS 7
#define BYTES 4
#define M 32
#define MAX_COMPONENTS 1000

struct Component {
    int type;
    int row;
    int column;
    int outerEdges[1000];
    int outerEdgesCount;
};
```

```
int readNumTemplates(FILE *input_file) ;

int validateCommandLineArguments(int argc) {
    if (argc != 4) {
        printf("Invalid arguements!\n");
        return 0; // Returning 0 to indicate an error
    }
    return 1; // Returning 1 to indicate success
}
```

```
int readNumTemplates(FILE *input_file) {
  // Read the number of templates from the file
  unsigned char tempNum;
  size_t bytesRead = fread(&tempNum, sizeof(unsigned char), 1, input_file);
  if (bytesRead != 1) {
    // Error reading the number of templates
    printf("Template number error.\n");
    return -1;
  }

  // Return the number of templates
  return tempNum;
}
```

```
int readTemplateData(FILE *file, int index, unsigned char *templateData) {
  // Seek to the beginning of the template data
  fseek(file, 1 + index * 128, SEEK_SET);

  // Read the template data from the file
  size_t bytesRead = fread(templateData, sizeof(unsigned char), 128, file);
  // for(int i =0; i<bytesRead; i++){
  //   printf("%d", templateData[i]);
  // }
  if (bytesRead != 128) {
    // Error reading the template data
    printf("Reading template error.\n");
    return -1;
  }

  // Return success
  return 0;
}

void printTemplateData(const unsigned char *templateData) {
  // Declare an array to store the template data
  unsigned char templateArray[32][BYTES];

  // Copy the template data to the array
  for (int row = 0; row < 32; row++) {
    for (int current_byte = 0; current_byte < BYTES; current_byte++) {
```

```
    templateArray[row][current_byte] = templateData[row * BYTES + current_byte];
   }
  }

  // Print the template data from the array
  printf("Template data:\n");
  for (int row = 31; row >= 0; row--) {
   for (int current_byte = 0; current_byte < BYTES; current_byte++) {
    unsigned char rowData = templateArray[row][current_byte];
    for (int current_byte = BITS; current_byte >= 0; current_byte--) {
     if (rowData & (1 << current_byte)) {
      printf("1");
     } else {
      printf(" ");
     }
    }
   }
   printf("\n");
  }
}

void detectTheshold(uint8_t*** pixels, unsigned int width, unsigned int height){
   int blue,green,red, Th ;
  for(unsigned int i = 0; i<height; i++){
   for(unsigned int j = 0; j<width; j++){
    blue =  pixels[i][j][0];
    red = pixels[i][j][1];
    green = pixels[i][j][2];
```

```
    //Th = sqrt(blue*blue + red*red + green * green);
    Th = (green + red + blue) / 3;
    if(Th >= 127){
     pixels[i][j][3] = 1;

    }
    else{
     pixels[i][j][3] = 0;
    }
    }
  }
  printf("Image height: %d, Image width: %d", height, width);
  for(unsigned int i = 0; i<height; i++){
    for(unsigned int j = 0; j<width; j++){
       if(pixels[i][j][3] == 1)
       printf("1");
       else
       printf(" ");
    }
    printf("\n");
    }
}



// Helper function for recursive depth-first search

bool dfs(struct Component *components, int i, int j, int targeti, int targetj, int width, int height, uint8_t
***pixels, bool visited[1000][1000], int counter, int pos) {
```

```
//Check boundary and if already visited and if reagion is true
 // for(int s = 0; s < 420; s ++)
 //    printf("%d,", pixels[286][s][3]);
//  printf("targeti:%d,targetj: %d,width: %d,height: %d\n", targeti, targetj, width, height);
//  printf("Recusion: i:%d, j:%d, value:%d, visited:%d\n", i, j, pixels[j][i][3],visited[i][j]);
//  getchar();
    if (i < 0 || i >= width || j < 0 || j >= height) {
  //    printf("Triggered\n");
      return false;
    }
visited[i][j] = true;

//create bound box
// Check for true condition
 if((abs(targeti - i) <= 32) && targeti < i && targetj == j)
 {//printf("C1");
  return true; }
 if((abs(targeti - i) <= 32)  && targeti < i && targetj+32 == j)
  { //printf("C2");
  return true; }
 if((abs(targetj-j) <= 32) && targetj < j && (targeti == i))
 {  //printf("C3");
     return true;}
 if((abs(targetj-j) <= 32) && targetj < j  && (targeti+32 == i))
  { // printf("C4");
  return true;}

for(int s = 0; s < counter; s++ )
```

```
    {
     if(s != pos )
     {
       if((abs(components[s].column-i) <= 32) && components[s].column < i && components[s].row == j)
         return false;
       if((abs(components[s].column - i) <= 32) && components[s].column < i  && components[s].row+32
== j)
         return false;
       if((abs(components[s].row-j) <= 32) && components[s].row < j  && (components[s].column == i))
         return false;
       if((abs(components[s].row-j) <= 32) && components[s].row < j && (components[s].column+32 == i))
         return false;
     }
   }



   // Check if connected to some other block too
  // printf("Check DONE");
   // Mark the current pixel as visited


   // Recursive DFS on neighbors
   int connected = 0;
  if(pixels[j+1][i][3] == 1 && !visited[i][j+1])
   connected += dfs(components, i, j + 1, targeti, targetj, width, height, pixels, visited, counter, pos); //
Right
  if(pixels[j-1][i][3] == 1 && !visited[i][j-1])
   connected += dfs(components, i, j - 1, targeti, targetj, width, height, pixels, visited, counter, pos); //
Left
  if(pixels[j][i+1][3] == 1 && !visited[i+1][j])
```

```
   connected += dfs(components, i + 1, j, targeti, targetj, width, height, pixels, visited, counter, pos); // Down

  if(pixels[j][i-1][3] == 1 && !visited[i-1][j])

   connected += dfs(components, i - 1, j, targeti, targetj, width, height, pixels, visited, counter, pos); // Up


   return connected>0;

}


// Function to check if two components are connected

bool areComponentsConnected(struct Component *components, struct Component component1, struct Component component2, int width, int height, uint8_t ***pixels, int counter, int pos) {

      bool visited[1000][1000] = {false};

      int i = component1.column;

      int j = component1.row;

      int targeti = component2.column;

      int targetj = component2.row;


        // If a connection is found, return true

      return (dfs( components, i, j, targeti, targetj, width, height, pixels, visited, counter, pos)) ;


}


int main(int argc, char *argv[]) {


   int size = 100; // assuming Maximum of 100 components in a PCB

   struct Component components[size];


   // Check the number of command line arguments
```

```
if (!validateCommandLineArguments(argc)) {
    return 1; // Exit the program with an error code
}

// Get the command line arguments
char *mode = argv[1];

if (strcmp(mode, "t") != 0 && strcmp(mode, "l") != 0 && strcmp(mode, "c") != 0) {
    printf("Invalid mode selected!\n");
    return 1;
}

if (strcmp(mode, "t") == 0) {
    // Read the template library file

    char *file_name = argv[2];
    int index = atoi(argv[3]);

    FILE *file = fopen(file_name, "rb");
    if (file == NULL) {
        printf("Could not open the file %s\n", file_name);
        return 1;
    }

    int TempNum = readNumTemplates(file);

    if (TempNum < 0 || index < 0 || index >= TempNum) {
        printf("Template index out of range\n");
```

```
        fclose(file);
        return 1;
    }

    unsigned char templateData[128];
    if (readTemplateData(file, index, templateData) < 0) {
        fclose(file);
        return 1;
    }

    fclose(file);

    printTemplateData(templateData);



} else if (strcmp(mode, "l")==0 || strcmp(mode,"c") == 0) {
    char *file_name = argv[2];
    char *imageFileName = argv[3];

    //Read templete File
    FILE *file = fopen(file_name, "rb");
    if (file == NULL) {
        printf("Could not open the file %s\n", file_name);
        return 1;
    }

    int TempNum = readNumTemplates(file);
```

```
//Read BMP file
Bmp image = read_bmp(imageFileName);
detectTheshold(image.pixels, image.width, image.height);
uint8_t templatePixel = 0;
int result_array[100][3] ;
int counter = 0;

int isMatch = 0;
char text_string[10000]={};
for(int row = 0 ; row < image.height - M; row++)
{
   for(int index = 0; index < TempNum; index++)
   {
      unsigned char templateData[128];
      if (readTemplateData(file, index, templateData) < 0) {
         fclose(file);
         return 1;
      }
      isMatch = 1;  // Assume it's a match
      for(int col = 0 ; col < image.width - M; col++)
      {

         for (int i = M-1; i >= 0; i--) {
            for (int j = 0; j < M ; j++) {
            // Compare the pixel values

               int byteIndex = j / 8;
               int bitIndex =  j % 8;
```

```
                templatePixel = (templateData[i * 4 + byteIndex] >> 7 - bitIndex) & 0x1;
                isMatch++;
                if(image.pixels[row+i][col+j][3] !=  templatePixel) {
                    isMatch = 0;  // Not a match
                    break;
                }
            }
            if(!isMatch)
                break;
        }
        if (isMatch)
        {
            // Report the match location
            components[counter].type = index;
            components[counter].row = row;
            components[counter].column = col;
            counter++;
        }
    }
  }
}
fclose(file);
if(counter > 0)
   printf("Found %d components:\n",counter);
for(int i = 0; i< counter; i++)
{
```

```
        printf("type: %d, row: %d, column: %d\n", components[i].type, components[i].row,
components[i].column);

      }

     if(strcmp(mode, "c") == 0 )

     {

      bool alone = true;

//        printf("DFS mode now\n");

          for(int i = 0; i < counter; i ++)

          {

            printf("Component %d connected to", i);

            alone = true;

            for(int j = 0; j < counter; j++)

            {

              if(i != j)

              {

                 if (areComponentsConnected(components, components[i], components[j], image.width,
image.height, image.pixels, counter, i)) {

                    printf(" %d", j);

                    alone = false;

                }


              }

            }

            if(alone == false)

              printf("\n");

            else

              printf(" nothing\n");

          }
```

```
    }
    }
    else {
        printf("Invalid mode selected!\n");
        return 1;

    }

    return 0;
}
```