```cpp
/* ICC Pratique - CS-119
 * Mini-Projet  - ColoReduce
 *
 * Auteur: Rafael Riber
 * Numéro SCIPER: 296142
 * Date:  Décembre 2018
 *
 * Ce programme prend en entrée un fichier contenant des instructions de
 * normalisation (nombre et valeurs RGB de couleurs réduites, seuils de
 * normalisation) et d'un nombre F de filtrages à effectuer, ainsi qu'une image au
 * format PPM, et renvoie cette image, réduite aux couleurs données en entrée, puis
 * filtrée F fois.
*/

#include <iostream>
#include <vector>
#include <cmath>
#include <string>
using namespace std;

const int maxVal(255);              // Intensité maximale d'une composante de couleur
const int minNbR(2);                // Nombre minimum de couleurs réduites
const int maxNbR(255);              // Nombre maximum de couleurs réduites

const int filterColor(0);           // Indice de couleur utilisé dans le filtrage
                                    // dans le cas ou on ne trouverait pas 6 voisins
                                    // de la même valeur

const int maxSameVals(6);           // Condition d'arrêt pour le filtrage
const double epsilon(0.001);        // Utile dans la vérification des seuils
const double firstThreshold(0.0);   // Premier seuil implicite
const double lastThreshold(1.0);    // Dernier seuil implicite

struct Color;

typedef vector<vector<int>>   NormImg;
typedef vector<vector<Color>> RGBImg;
typedef vector<Color>         ReducedColors;
typedef vector<double>        ThresholdList;
typedef vector<int>           NeighborCounter;

struct Color{
    int r, g, b;
};

//Décomposition du fichier d'entrée
struct InputImg{
    int nbR;                    // Nombre de couleurs réduites
    int nbF;                    // Nombre de filtrages
    unsigned int nbC;           // Nombre de pixels horizontaux
    unsigned int nbL;           // Nombre de pixels verticaux
    ReducedColors rColors;      // Liste des couleurs réduites
    ThresholdList thresholds;   // Liste des seuils
    RGBImg inputImg;            // Pixels de l'image d'entrée
};

// Première couleur réduite
const Color black = {0,0,0};  // Première couleur réduite constante

// Fonctions d'erreur
void error_nbR(int nbR);
void error_color(int id);
void error_threshold(double invalid_val);
void error_nb_filter(int nb_filter);


```

```
67 // Fonctions de lecture
68 InputImg fileRead();
69 Color colorRead();
70 void inputReduced(InputImg& input);
71 void inputThresholds(InputImg& input);
72 void inputFilters(InputImg& input);
73 void inputDimensions(InputImg& input);
74 void inputPixels(InputImg& input);
75
76 // Fonctions de transformation
77 NormImg normalize(InputImg rgb);
78 void filter(NormImg& source, int nbL, int nbC, int nbF, int nbR);
79 int getPixelValue(int x, int y, int nbR, NormImg& source);
80 void blackEdge(NormImg& norm, int nbL, int nbC, int nbF);
81
82 // Fonction de rendu
83 RGBImg render(NormImg filtered, int nbL, int nbC, ReducedColors rColors);
84 void   printRGB(RGBImg rgb, int nbL, int nbC);
85
86 int main()
87 {
88     // On lit le fichier d'entrée, et on le stocke dans la structure "image"
89     InputImg image = fileRead();
90
91     // On seuille l'image d'entrée, et on stocke le résultat dans un tableau "norm'
92     NormImg norm = normalize(image);
93
94     // On filtre l'image normalisée
95     filter(norm, image.nbL, image.nbC, image.nbF, image.nbR);
96
97     // On crée une image RGB résultat à partir des couleurs réduites et de l'image
98     // filtrée
99     RGBImg rendered = render(norm, image.nbL, image.nbC, image.rColors);
100
101    // On imprime l'image RGB résultat correctement formatée au format PPM
102    printRGB(rendered, image.nbL, image.nbC);
103
104    return 0;
105 }
106
107 void error_nbR(int nbR)
108 {
109     cout << "Invalid number of colors: " << nbR << endl;
110 }
111
112 void error_color(int id)
113 {
114     cout << "Invalid color value " << id << endl;
115 }
116
117 void error_threshold(double invalid_val)
118 {
119     cout << "Invalid threshold value: " << invalid_val << endl;
120 }
121
122 void error_nb_filter(int nb_filter)
123 {
124     cout << "Invalid number of filter: " << nb_filter << endl;
125 }
126
127 InputImg fileRead(){
128     InputImg input;
129
130     // Entrée des couleurs réduites
131     inputReduced(input);
132
```

```
133     // Entrée des seuils
134     inputThresholds(input);
135 
136     // Entrée des filtres
137     inputFilters(input);
138 
139     // L'en-tête ne nous intéresse pas
140     string header;
141     cin >> header;
142 
143     // Entrée des dimensions
144     inputDimensions(input);
145 
146     // La valeur maximale ne nous intéresse pas
147     int m;
148     cin >> m;
149 
150     // Pixels de l'image d'entrée
151     inputPixels(input);
152 
153     return input;
154 }
155 
156 void inputReduced(InputImg& input){
157     int n(0);
158 
159     cin >> n;
160     if(n < minNbR or n > maxNbR){
161         error_nbR(n);
162         exit(0);
163     }
164     input.nbR = n;
165 
166     // La première couleur est toujours le noir
167     input.rColors.push_back(black);
168 
169     for (int i(1); i < n+1; i++){
170         Color p(colorRead());
171 
172         if(p.r < 0 or p.r > maxVal){
173             error_color(i);
174             exit(0);
175         }
176         if(p.g < 0 or p.g > maxVal){
177             error_color(i);
178             exit(0);
179         }
180         if(p.b < 0 or p.b > maxVal){
181             error_color(i);
182             exit(0);
183         }
184 
185         input.rColors.push_back(p);
186     }
187 }
188 
189 void inputThresholds(InputImg& input){
190 
191     input.thresholds.push_back(firstThreshold);  // Le premier seuil est fixe
192 
193     for (int i(1); i < input.nbR; i++){
194         double t(0);
195         cin >> t;
196 
197         // Vérification d'écart entre le seuil actuel et le précédent
198         double deltaThresholds(abs(t - input.thresholds[i-1]));
```

```
199
200         if(deltaThresholds < epsilon){
201             error_threshold(t);
202             exit(0);
203         }
204         if(t < input.thresholds[i-1]){
205             error_threshold(t);
206             exit(0);
207         }
208
209         input.thresholds.push_back(t);
210     }
211     input.thresholds.push_back(lastThreshold);    // Le dernier seuil est fixe
212 }
213
214 void inputFilters(InputImg& input){
215     cin >> input.nbF;
216     if (input.nbF < 0) {
217         error_nb_filter(input.nbF);
218         exit(0);
219     }
220 }
221
222 void inputDimensions(InputImg& input){
223     cin >> input.nbC;
224     cin >> input.nbL;
225 }
226
227 void inputPixels(InputImg& input){
228     unsigned int l(input.nbL);
229     unsigned int c(input.nbC);
230
231     input.inputImg.resize(l);
232
233     for (auto &i : input.inputImg) {
234         for (size_t j(0); j < c; j++) {
235
236             Color p(colorRead());
237
238             if (p.r < 0 or p.r > maxVal) {
239                 error_color(p.r);
240                 exit(0);
241             }
242             if (p.g < 0 or p.g > maxVal) {
243                 error_color(p.g);
244                 exit(0);
245             }
246             if (p.b < 0 or p.b > maxVal) {
247                 error_color(p.b);
248                 exit(0);
249             }
250
251             i.push_back(p);
252         }
253     }
254 }
255
256 // Lecture d'une couleur
257 Color colorRead(){
258     Color p = black;
259     cin >> p.r;
260     cin >> p.g;
261     cin >> p.b;
262     return p;
263 }
264
```

```
265 NormImg normalize(InputImg rgb){
266     int l = rgb.nbL;
267     int c = rgb.nbC;
268     NormImg normOut(l,vector<int>(c));
269 
270     int nbR(rgb.nbR);
271 
272     for (unsigned int i(0); i < rgb.nbL; i++){
273         for (unsigned int j(0); j < rgb.nbC; j++){
274 
275             int r(rgb.inputImg[i][j].r);
276             int g(rgb.inputImg[i][j].g);
277             int b(rgb.inputImg[i][j].b);
278 
279             // Calcul de l'intensité normalisée
280             double normInt(sqrt(r*r + g*g + b*b) / (sqrt(3) * maxVal));
281 
282             for (int k(0); k <= nbR; k++){
283                 if (k == nbR && normInt >= rgb.thresholds[k-1]){
284                     normOut[i][j] = nbR;
285                 }
286                 if (normInt >= rgb.thresholds[k-1] && normInt < rgb.thresholds[k]){
287                     normOut[i][j] = k;
288                 }
289             }
290         }
291     }
292     return normOut;
293 }
294 
295 void filter(NormImg& source, int nbL, int nbC, int nbF, int nbR) {
296 
297     NormImg destination = source;
298     int val(0);
299 
300     for (int n(1); n <= nbF; n++){
301         for (int x(1); x < nbL-1; x++) {
302             for (int y(1); y < nbC-1; y++){
303                 val = getPixelValue(x, y, nbR, source);
304                 destination[x][y] = val;
305             }
306         }
307         source = destination;
308     }
309 
310 //  Bordure noire
311     blackEdge(source, nbL, nbC, nbF);
312 }
313 
314 int getPixelValue(int x, int y, int nbR, NormImg& source){
315 
316     NeighborCounter count(nbR + 1);
317     int current(0);
318 
319     // On itère parmi les voisins du pixel de coordonnées (x,y)
320     for (int i(-1); i <= 1; i++) {
321         for (int j(-1); j <= 1; j++) {
322 
323             if (i != 0 or j != 0) {
324                 // On stocke la valeur d'un voisin
325                 current = source[x + i][y + j];
326 
327                 // On compte le nombre de voisins identiques dans une liste "count'
328                 for (int c(0); c <= nbR; c++) {
329                     if (c == current) {
330 
```

```
                        count[c] = count[c] + 1;

                        // Condition d'arrêt si on a 6 voisins de même valeur
                        if (count[c] >= maxSameVals) {
                            return c;
                        }
                    }
                }
            }
        }
    }
    return filterColor;
}

void blackEdge(NormImg& norm, int nbL, int nbC, int nbF) {
    if (nbF > 0) {
        for (int i(0); i < nbL; i++) {
            for (int j(0); j < nbC; j++) {

                // Si le pixel est en bordure, on modifie
                if (i == 0 or j == 0 or i == nbL - 1 or j == nbC - 1) {
                    norm[i][j] = filterColor;
                }
            }
        }
    }
}

RGBImg render(NormImg filtered, int nbL, int nbC, ReducedColors rColors){
    RGBImg rendered;

    RGBImg inputImg(nbL,vector<Color>(nbC));
    rendered = inputImg;

    for (int i(0); i < nbL; i++){
        for (int j(0); j < nbC; j++){

            int normPixelVal(filtered[i][j]);

            rendered[i][j] = rColors[normPixelVal];
        }
    }
    return rendered;
}

void printRGB(RGBImg rgb, int nbL, int nbC){
    cout << "P3" << endl;
    cout << nbC << " " << nbL << endl;
    cout << maxVal << endl;
    for (int i(0); i < nbL; i++){
        for (int j(0); j < nbC; j++){
            cout << rgb[i][j].r << " ";
            cout << rgb[i][j].g << " ";
            cout << rgb[i][j].b << " ";
        }
        cout << endl;
    }
    cout << endl; // Pour que le résultat soit identique à celui de la demo
}
```