```c
typedef struct fonction{
    int *L;
    int **valeurs;
} fonction;

typedef struct structurant{
    bool **forme;
    int *origine;
    int rayon;
} structurant;

typedef struct image{
    int hauteur;
    int largeur;
    int **valeurs;
} image;

void ecrire_image(char *nom_fichier, image *img){
    FILE *fichier = fopen(nom_fichier, "w");
    if(fichier == NULL){
        printf("Erreur ! Fichier non ouvert.");
        return;
    }
    fprintf(fichier, "P2\n%d %d\n", img->largeur, img->hauteur);
    fprintf(fichier, "%d\n", 255);
    for(int i = 0; i < img->hauteur; i++){
        for(int j = 0; j < img->largeur; j++){
            fprintf(fichier, "%d ", img->valeurs[i][j]);
        }
        fprintf(fichier, "\n");
    }
    fclose(fichier);
}

image *recupere_image(char *nom_fichier){
    FILE *fichier = fopen(nom_fichier, "r");
    if(fichier == NULL){
        printf("Erreur ! Fichier non ouvert.\n");
        return NULL;
    }
    char *mode = malloc(3 * sizeof(char));
    if (mode == NULL) {
        printf("Erreur d'allocation pour le mode.\n");
        fclose(fichier);
        return NULL;
    }

    int hauteur, largeur, max;
    fscanf(fichier, "%s %d %d %d", mode, &hauteur, &largeur, &max);
    printf("%s -> Mode: %s, Largeur: %d, Hauteur: %d, Max: %d\n", nom_fichier,
   mode, largeur, hauteur, max);


    image *img = malloc(sizeof(image));
    if (img == NULL) {
        printf("Erreur d'allocation pour l'image.\n");
```

```
56          free(mode);
57          fclose(fichier);
58          return NULL;
59      }
60
61      img->hauteur = hauteur;
62      img->largeur = largeur;
63      img->valeurs = malloc(hauteur * sizeof(int*));
64      if (img->valeurs == NULL) {
65          printf("Erreur d'allocation pour les lignes.\n");
66          free(mode);
67          free(img);
68          fclose(fichier);
69          return NULL;
70      }
71
72      for(int i = 0; i < hauteur; i++){
73          img->valeurs[i] = malloc(largeur * sizeof(int));
74          if (img->valeurs[i] == NULL) {
75              printf("Erreur d'allocation pour la colonne %d.\n", i);
76              for (int k = 0; k < i; k++) free(img->valeurs[k]);
77              free(img->valeurs);
78              free(img);
79              free(mode);
80              fclose(fichier);
81              return NULL;
82          }
83      }
84
85      for(int i = 0; i < img->hauteur; i++){
86          for(int j = 0; j < img->largeur; j++){
87              fscanf(fichier, "%d", &(img->valeurs[i][j]));
88          }
89      }
90
91      free(mode);
92      fclose(fichier);
93      return img;
94  }
95
96  image *image_depuis_fonction(fonction *f){
97      image *img = malloc(sizeof(image));
98      img->hauteur = f->L[0];
99      img->largeur = f->L[1];
100     img->valeurs = malloc(img->hauteur * sizeof(int *));
101     for(int i = 0; i < img->hauteur; i++){
102         img->valeurs[i] = malloc(img->largeur * sizeof(int));
103         for(int j = 0; j < img->largeur; j++){
104             img->valeurs[i][j] = f->valeurs[i][j];
105         }
106     }
107     return img;
108 }
109
110 fonction *fonction_nulle(int *L){
111     fonction *f_0 = malloc(sizeof(fonction));
```

```
    f_0->L = L;
    f_0->valeurs = malloc(L[0]*sizeof(int*));
    for(int i = 0; i < L[0]; i++){
        f_0->valeurs[i] = malloc(L[1]*sizeof(int));
        for(int j = 0; j < L[1]; j++){
            f_0->valeurs[i][j] = 0;
        }
    }
    return f_0;
}

structurant *structurant_vide(int *origine, int rayon){
    structurant *ret = malloc(sizeof(structurant));
    ret->rayon = rayon;
    ret->forme = malloc(2*rayon*sizeof(bool*));
    for(int i = 0; i < 2*rayon; i++){
        ret->forme[i] = calloc(2*rayon, sizeof(bool));
    }
    ret->origine = origine;
    return ret;
}

void dessine_fonction(fonction *f, fonction *g){
    for(int i = 0; i < f->L[0]; i++){
        for(int j = 0; j < f->L[1]; j++){
            if(g->valeurs[i][j] > 0){
                f->valeurs[i][j] = g->valeurs[i][j];
            }
        }
    }
    return;
}

void libere_fonction(fonction *f){
    for(int i = 0; i < f->L[0]; i++){
        free(f->valeurs[i]);
    }
    free(f->valeurs);
    free(f);
    return;
}

void libere_structurant(structurant *B){
    for(int i = 0; i < 2*B->rayon; i++){
        free(B->forme[i]);
    }
    free(B->forme);
    free(B);
    return;
}

void libere_image(image *img){
    for(int i = 0; i < img->hauteur; i++){
        free(img->valeurs[i]);
    }
    free(img->valeurs);
```

```
168     free(img);
169     return;
170 }
171 
172 fonction *fonction_of_image(image *img){
173     int *L = malloc(2*sizeof(int));
174     L[0] = img->hauteur; L[1] = img->largeur;
175     fonction *C = fonction_nulle(L);
176     for(int i = 0; i < L[0]; i++){
177         for(int j = 0; j < L[1]; j++){
178             C->valeurs[i][j] = img->valeurs[i][j];
179         }
180     }
181     return C;
182 }
183 
184 fonction *fonction_of_structurant(structurant *B){
185     int *L = malloc(2*sizeof(int));
186     L[0] = 2*B->rayon;
187     L[1] = 2*B->rayon;
188     fonction *f = fonction_nulle(L);
189     f->L = L;
190     for(int i = 0; i < 2*B->rayon; i++){
191         for(int j = 0; j < 2*B->rayon; j++){
192             f->valeurs[i][j] = 255*B->forme[i][j];
193         }
194     }
195     return f;
196 }
197 
198 bool est_dedans(int i, int j, int *L){
199     return 0 <= i && i < L[0] && 0 <= j && j < L[1];
200 }
201 
202 int sup(fonction *f, structurant *B){
203     int s = 0;
204     int o[2] = {B->origine[0], B->origine[1]};
205     int r = B->rayon;
206     for(int i = o[0] - r; i < o[0] + r; i++){
207         for(int j = o[1] - r; j < o[1] + r; j++){
208             if(est_dedans(i, j, f->L) && B->forme[i + r - o[0]][j + r - o[1]]
    && s <= f->valeurs[i][j]){
209                 s = f->valeurs[i][j];
210             }
211         }
212     }
213     return s;
214 }
215 
216 int inf(fonction *f, structurant *B){
217     int s = 255;
218     int o[2] = {B->origine[0], B->origine[1]};
219     int r = B->rayon;
220     for(int i = o[0] - r; i < o[0] + r; i++){
221         for(int j = o[1] - r; j < o[1] + r; j++){
222             if(est_dedans(i, j, f->L) && B->forme[i + r - o[0]][j + r - o[1]]
```

```
    && s >= f->valeurs[i][j]){
                s = f->valeurs[i][j];
            }
        }
    }
    return s;
}

void translate_B_vers_x(structurant *B, int *x){
    B->origine = x;
    return;
}

structurant *Disque(int *L, int r, int *o){
    structurant *B = structurant_vide(o, r);
    for(int i = o[0] - r; i < o[0] + r; i++){
        for(int j = o[1] - r; j< o[1] + r; j++){
            int dist_2 = (i-o[0]) * (i-o[0]) + (j-o[1]) * (j-o[1]);
            if(est_dedans(i, j, L)){
                if(dist_2 <= r*r){
                    B->forme[i + r - o[0]][j + r - o[1]] = true;
                }
            }
        }
    }
    return B;
}

fonction *dilatation_fonctionnelle(fonction *f, structurant *B){
    fonction *f_d = fonction_nulle(f->L);
    int *x = malloc(2*sizeof(int));
    for(int i = 0; i < f->L[0]; i++){
        x[0] = i;
        for(int j = 0; j < f->L[1]; j++){
            x[1] = j;
            translate_B_vers_x(B, x);
            f_d->valeurs[i][j] = sup(f, B);
        }
    }
    free(x);
    return f_d;
}

fonction *erosion_fonctionnelle(fonction *f, structurant *B){
    fonction *f_e = fonction_nulle(f->L);
    int *x = malloc(2*sizeof(int));
    for(int i = 0; i < f->L[0]; i++){
        x[0] = i;
        for(int j = 0; j < f->L[1]; j++){
            x[1] = j;
            translate_B_vers_x(B, x);
            f_e->valeurs[i][j] = inf(f, B);
        }
    }
    free(x);
    return f_e;
```

```
}

fonction *ouverture_numerique(fonction *f, structurant *B){
    return dilatation_fonctionnelle(erosion_fonctionnelle(f, B), B);
}

fonction *fermeture_numerique(fonction *f, structurant *B){
    return erosion_fonctionnelle(dilatation_fonctionnelle(f, B), B);
}

fonction *rehaussement_contraste(fonction *f, double alpha, double beta,
   fonction *f_inf, fonction *f_sup){
    fonction *g = fonction_nulle(f->L);
    for(int i = 0; i < f->L[0]; i++){
        for(int j = 0; j < f->L[1]; j++){
            int Delta_f_i_j = f_sup->valeurs[i][j] - f_inf->valeurs[i][j];
            if(f_inf->valeurs[i][j] <= f->valeurs[i][j] && f->valeurs[i][j] <=
   f_inf->valeurs[i][j] + alpha*Delta_f_i_j){
                g->valeurs[i][j] = f_inf->valeurs[i][j];
            }else if(f_inf->valeurs[i][j] + alpha*Delta_f_i_j <= f->valeurs[i][
   j] && f->valeurs[i][j] <= f_sup->valeurs[i][j] - beta*Delta_f_i_j){
                g->valeurs[i][j] = f->valeurs[i][j];
            }else{
                g->valeurs[i][j] = f_sup->valeurs[i][j];
            }
        }
    }
    return g;
}

fonction *gradient_morpho(fonction *f, int *origine){
    fonction *g = fonction_nulle(f->L);
    structurant *B = Disque(f->L, 1, origine);
    fonction *D = dilatation_fonctionnelle(f, B);
    fonction *E = erosion_fonctionnelle(f, B);
    for(int i = 0; i < f->L[0]; i++){
        for(int j = 0; j < f->L[1]; j++){
            g->valeurs[i][j] = D->valeurs[i][j] - E->valeurs[i][j];
        }
    }
    return g;
}

fonction *chapeau_haut_de_forme(fonction *f, structurant *B){
    fonction *g = fonction_nulle(f->L);
    fonction *ouv = ouverture_numerique(f, B);
    for(int i = 0; i < f->L[0]; i++){
        for(int j = 0; j < f->L[1]; j++){
            if(f->valeurs[i][j] - ouv->valeurs[i][j] <= 0){
                g->valeurs[i][j] = 0;
            }
            else{
                g->valeurs[i][j] = f->valeurs[i][j] - ouv->valeurs[i][j];
            }
        }
    }
```

```
    return g;
}

int main(void){

    double alpha = 0.4;
    double beta = 0.5;
    int r_dilatation = 2;
    int r_erosion = 2;
    int r_ouverture = 2;
    int r_fermeture = 2;
    int r_chapeau = 2;

    image *img = recupere_image("Femme_P2.pgm");
    int L[2] = {img->hauteur, img->largeur};
    int origine[2] = {img->hauteur/2, img->largeur/2};
    fonction *f = fonction_of_image(img);
    structurant *D = Disque(L, 2, origine);

    fonction *f_inf = ouverture_numerique(f, D);
    fonction *f_sup = fermeture_numerique(f, D);
    fonction *f_dilatation = dilatation_fonctionnelle(f, D);
    fonction *f_erosion = erosion_fonctionnelle(f, D);
    fonction *f_ouverture = ouverture_numerique(f, D);
    fonction *f_fermeture = fermeture_numerique(f, D);
    fonction *f_rehaussement = rehaussement_contraste(f, alpha, beta, f_inf,
    f_sup);
    fonction *f_gradient = gradient_morpho(f, origine);
    fonction *f_chapeau = chapeau_haut_de_forme(f, D);

    ecrire_image("Femme_dilatee_r=2.pgm", image_depuis_fonction(f_dilatation));
    ecrire_image("Femme_erodee_r=2.pgm", image_depuis_fonction(f_erosion));
    ecrire_image("Femme_ouverte_r=2.pgm", image_depuis_fonction(f_ouverture));
    ecrire_image("Femme_fermee_r=2.pgm", image_depuis_fonction(f_fermeture));
    ecrire_image("Femme_rehaussee_ouv_fer_alpha=04_beta=05.pgm",
    image_depuis_fonction(f_rehaussement));
    ecrire_image("Femme_gradient.pgm", image_depuis_fonction(f_gradient));
    ecrire_image("Femme_chapeau_hdf_r=2.pgm", image_depuis_fonction(f_chapeau))
    ;

    libere_structurant(D);
    libere_fonction(f);
    libere_fonction(f_inf);
    libere_fonction(f_sup);
    libere_fonction(f_dilatation);
    libere_fonction(f_erosion);
    libere_fonction(f_ouverture);
    libere_fonction(f_fermeture);
    libere_fonction(f_rehaussement);
    libere_fonction(f_gradient);
    libere_fonction(f_chapeau);
    libere_image(img);

    return 0;
}
```