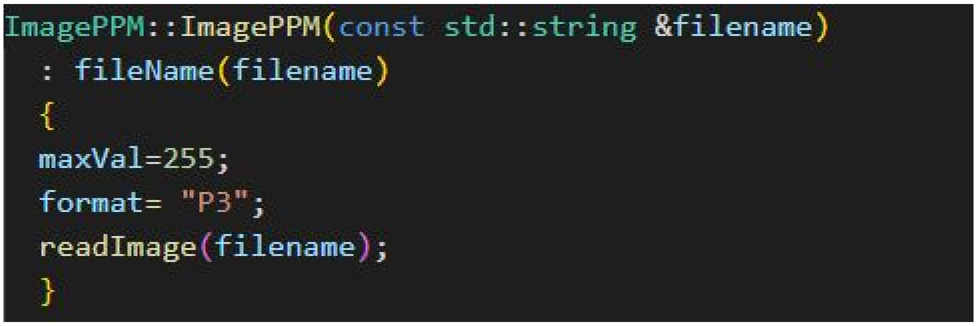
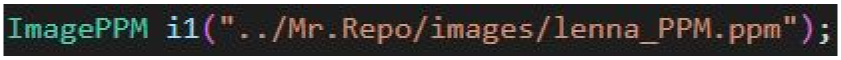
PRUEBAS PROCESAMIENTO DE IMÁGENES PPM Y PGM EN C++

Salazar, Ángelo., Revelo, Juan José., Bailon, Juan José. angelo.salazar@correounivalle.edu.co, juan.jose.revelo@correounivalle.edu.co, juan.bailon@correounivalle.edu.co
Universidad del Valle.

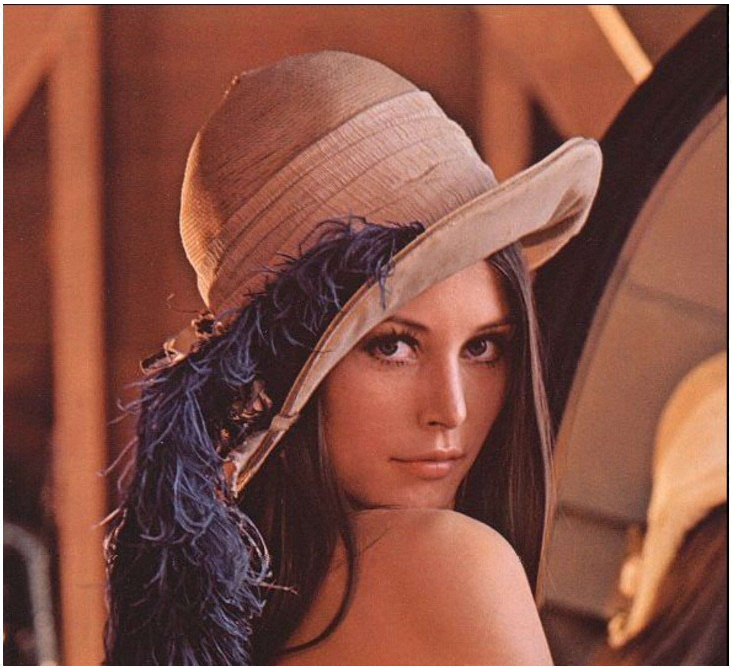
1. *Prueba lectura de imagen*



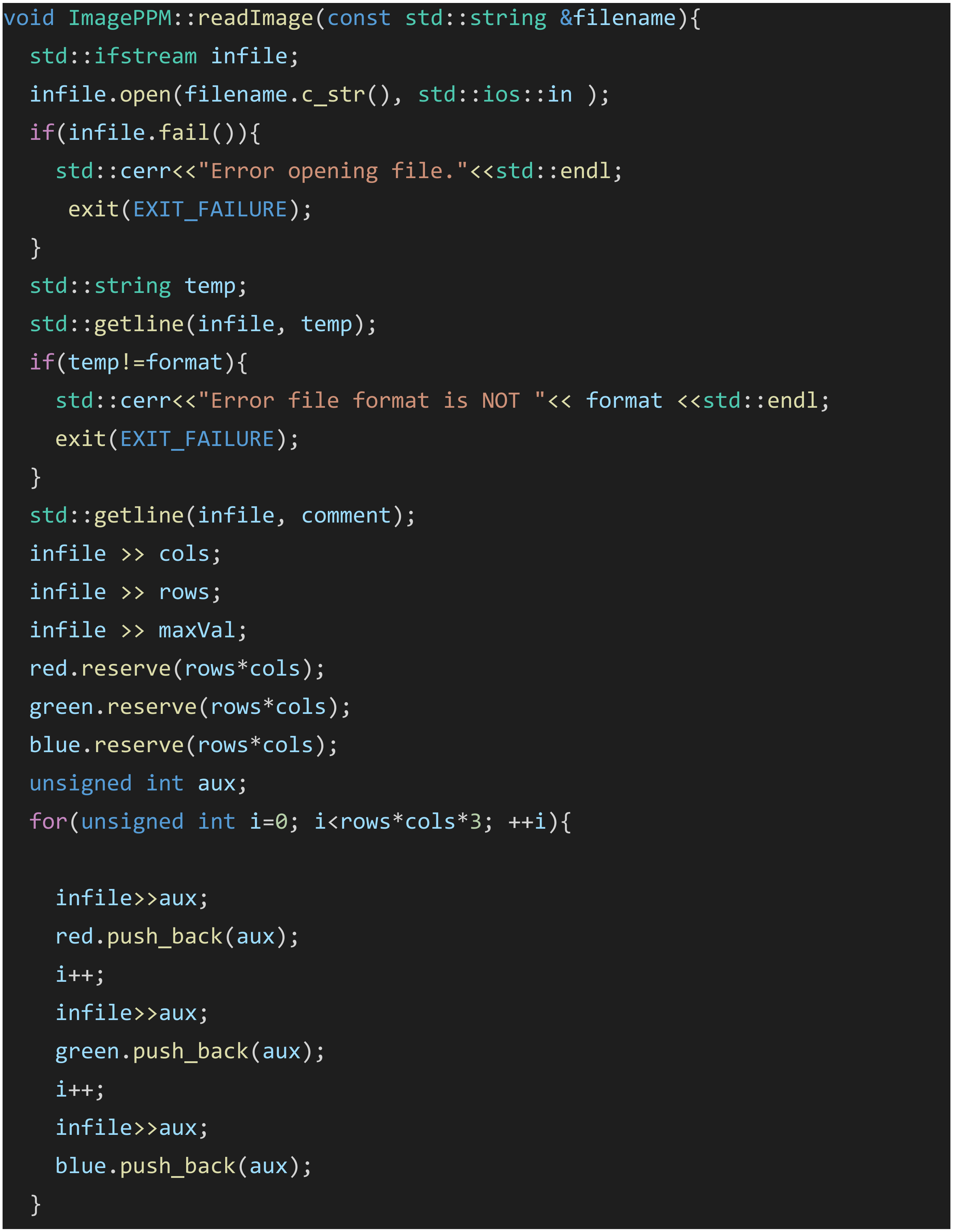
```
ImagePPM::ImagePPM(const std::string &filename)
  : fileName(filename)
  {
  maxVal=255;
  format= "P3";
  readImage(filename);
  }
```



```
ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm");
```



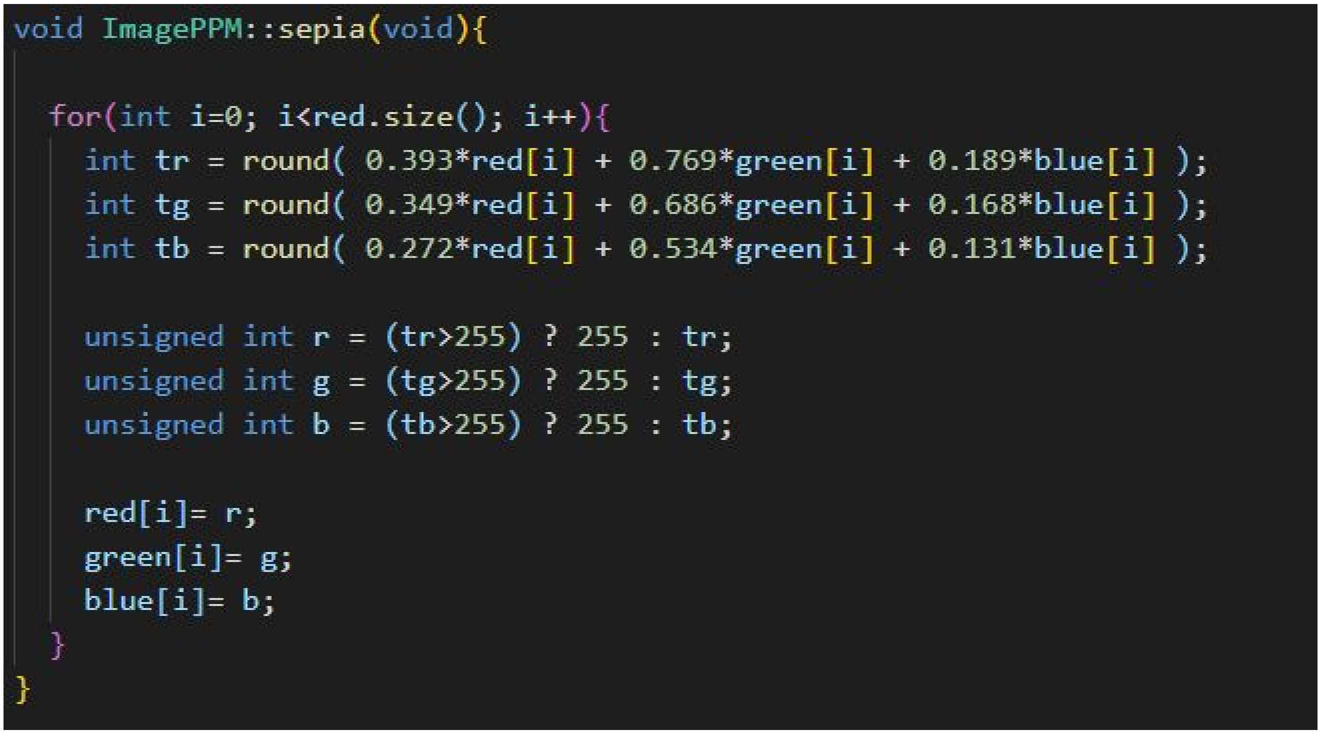
2. *Prueba escritura de imagen*



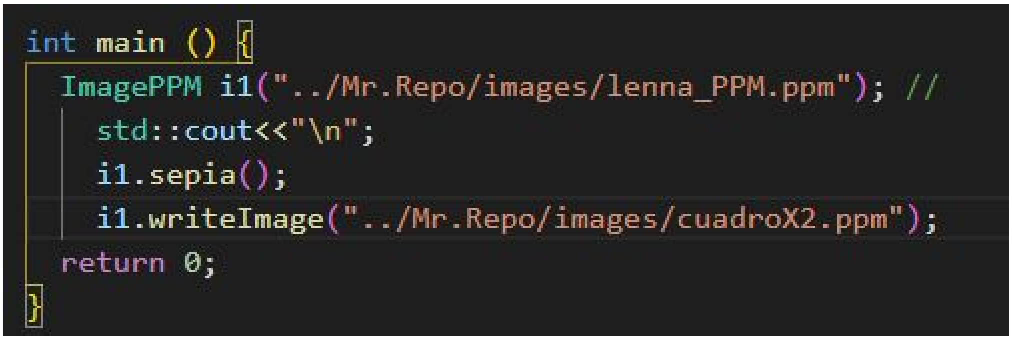
```cpp
void ImagePPM::readImage(const std::string &filename){
  std::ifstream infile;
  infile.open(filename.c_str(), std::ios::in );
  if(infile.fail()){
    std::cerr<<"Error opening file."<<std::endl;
     exit(EXIT_FAILURE);
  }
  std::string temp;
  std::getline(infile, temp);
  if(temp!=format){
    std::cerr<<"Error file format is NOT "<< format <<std::endl;
    exit(EXIT_FAILURE);
  }
  std::getline(infile, comment);
  infile >> cols;
  infile >> rows;
  infile >> maxVal;
  red.reserve(rows*cols);
  green.reserve(rows*cols);
  blue.reserve(rows*cols);
  unsigned int aux;
  for(unsigned int i=0; i<rows*cols*3; ++i){

    infile>>aux;
    red.push_back(aux);
    i++;
    infile>>aux;
    green.push_back(aux);
    i++;
    infile>>aux;
    blue.push_back(aux);
  }
  if(rows*cols != red.size() || rows*cols != green.size() || rows*cols != blue.size()){
    std::cout<<"Could not read all the pixels on the file."<<std::endl;
  }else{
    std::cout<<"Everything is good, could read all the pixels on the image.";
  }
  infile.close();
}
```

```
i1.writeImage("../Mr.Repo/images/cuadroX.ppm");
```

3. *Prueba filtro sepia*

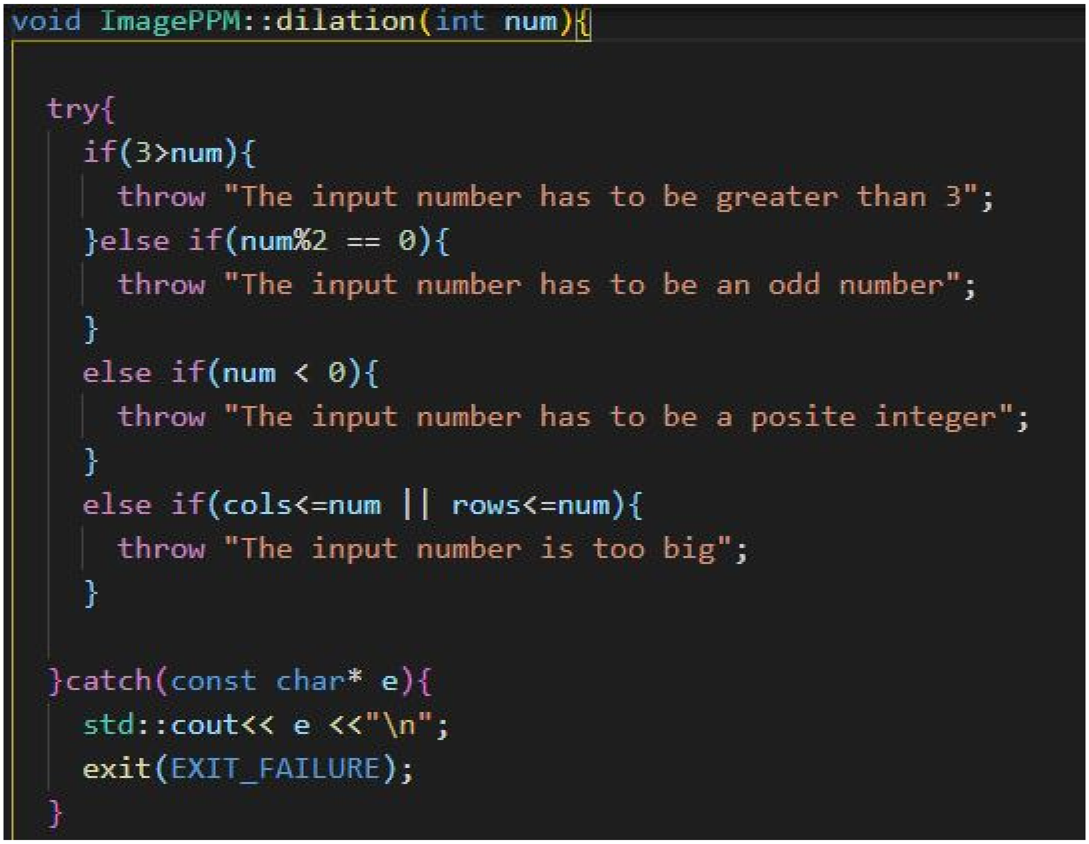


```
void ImagePPM::sepia(void){

  for(int i=0; i<red.size(); i++){
    int tr = round( 0.393*red[i] + 0.769*green[i] + 0.189*blue[i] );
    int tg = round( 0.349*red[i] + 0.686*green[i] + 0.168*blue[i] );
    int tb = round( 0.272*red[i] + 0.534*green[i] + 0.131*blue[i] );

    unsigned int r = (tr>255) ? 255 : tr;
    unsigned int g = (tg>255) ? 255 : tg;
    unsigned int b = (tb>255) ? 255 : tb;

    red[i]= r;
    green[i]= g;
    blue[i]= b;
  }
}
```

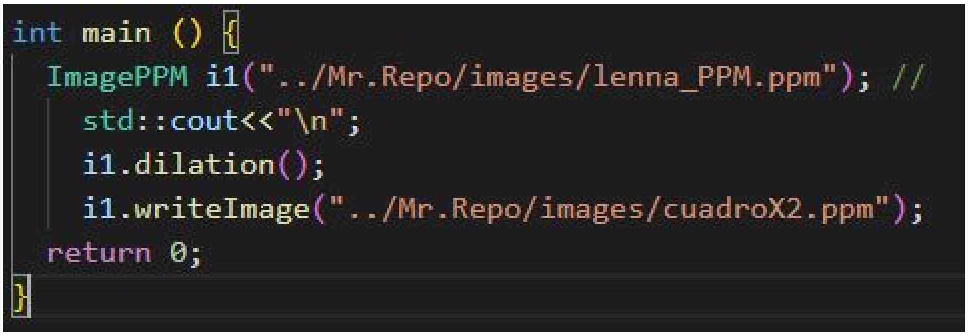


```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.sepia();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

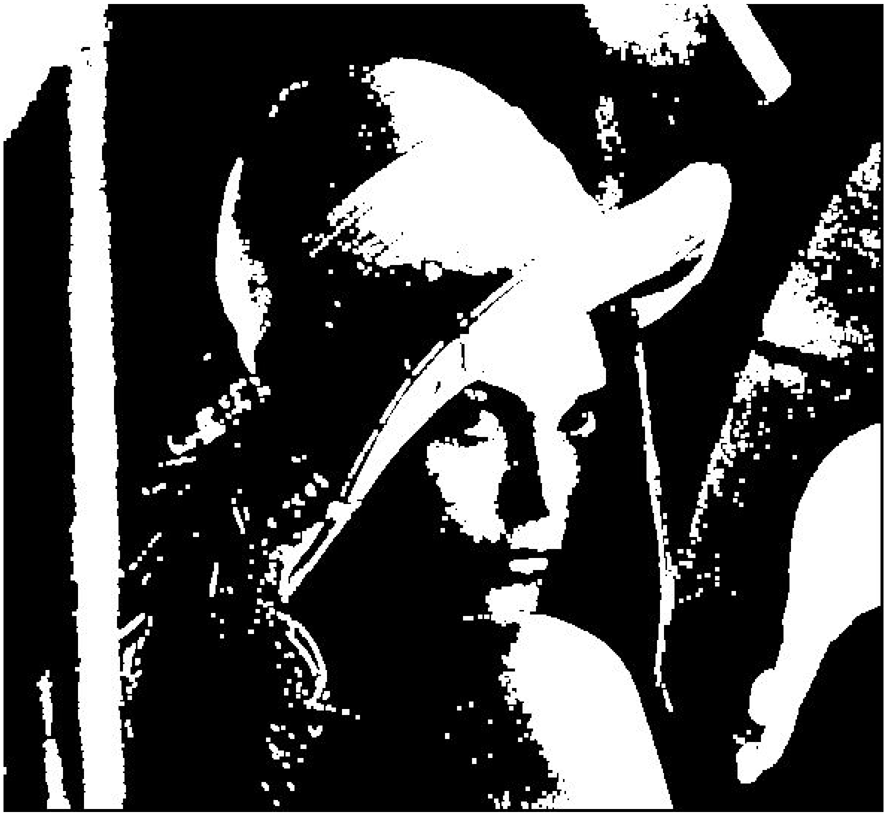
4. *Prueba filtro dilation*



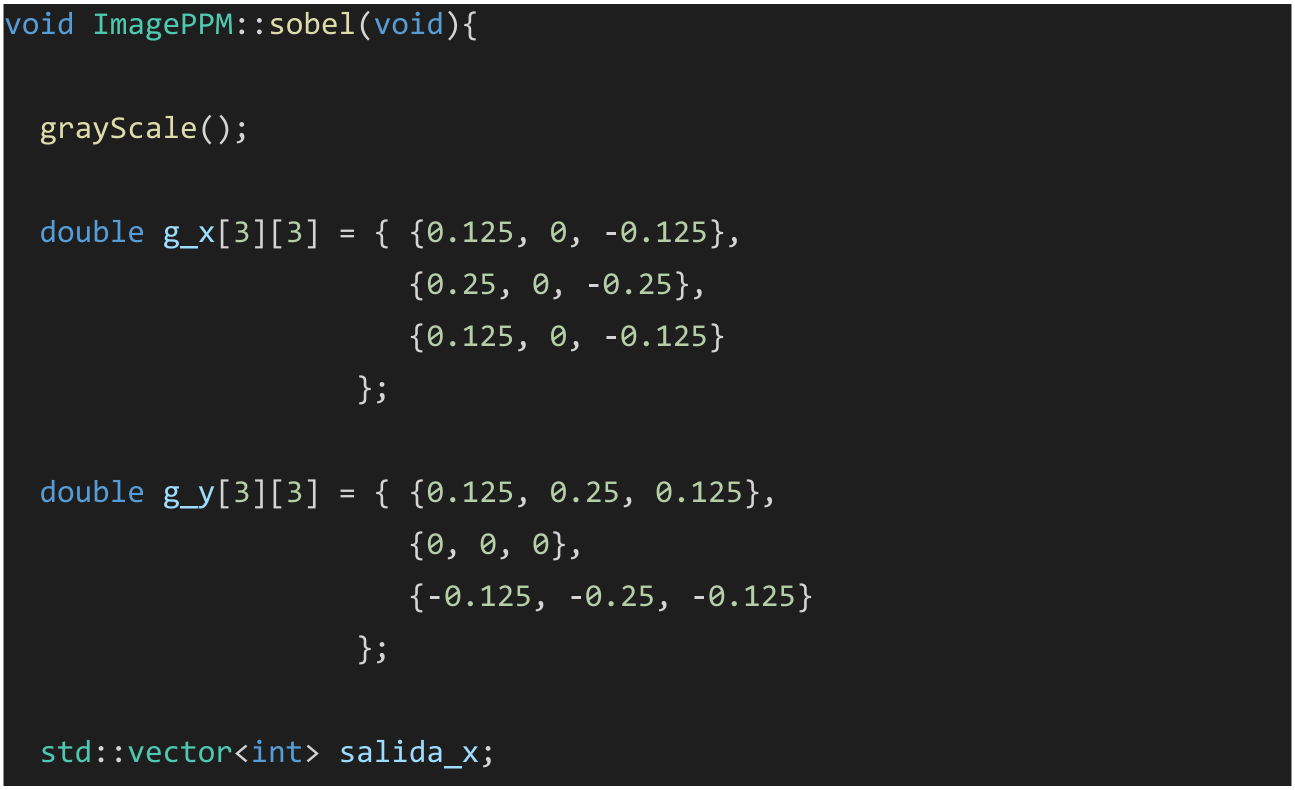
```
void ImagePPM::dilation(int num){

  try{
    if(3>num){
      throw "The input number has to be greater than 3";
    }else if(num%2 == 0){
      throw "The input number has to be an odd number";
    }
    else if(num < 0){
      throw "The input number has to be a posite integer";
    }
    else if(cols<=num || rows<=num){
      throw "The input number is too big";
    }

  }catch(const char* e){
    std::cout<< e <<"\n";
    exit(EXIT_FAILURE);
  }
```



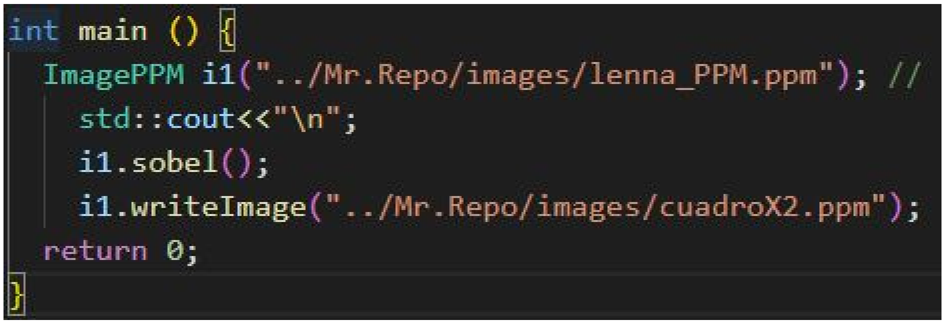
```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.dilation();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```



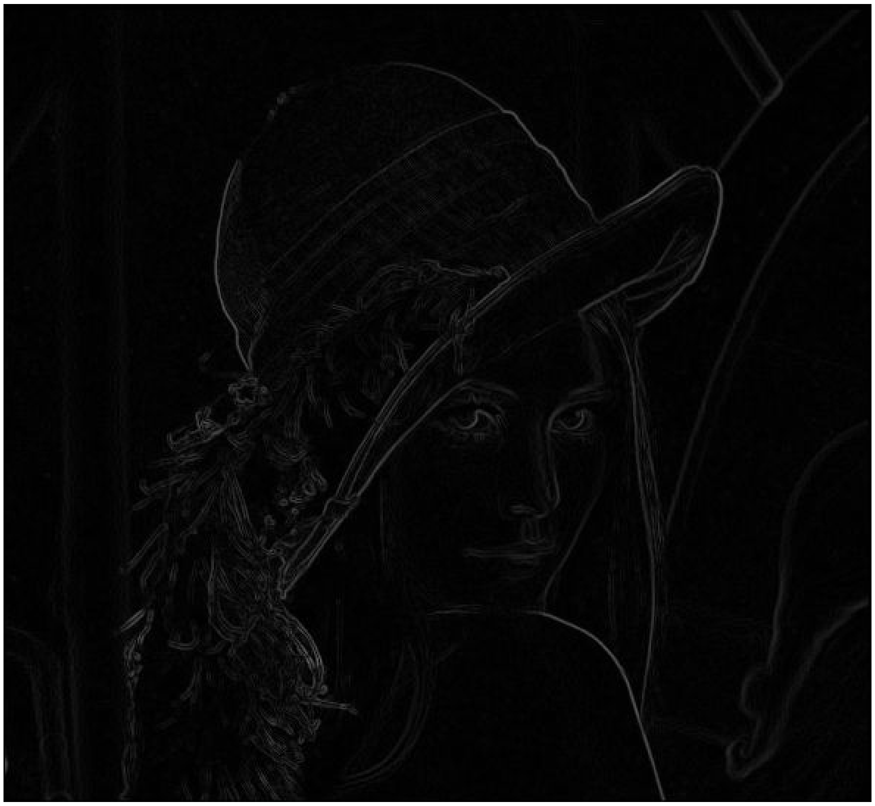
5. Prueba filtro sobel



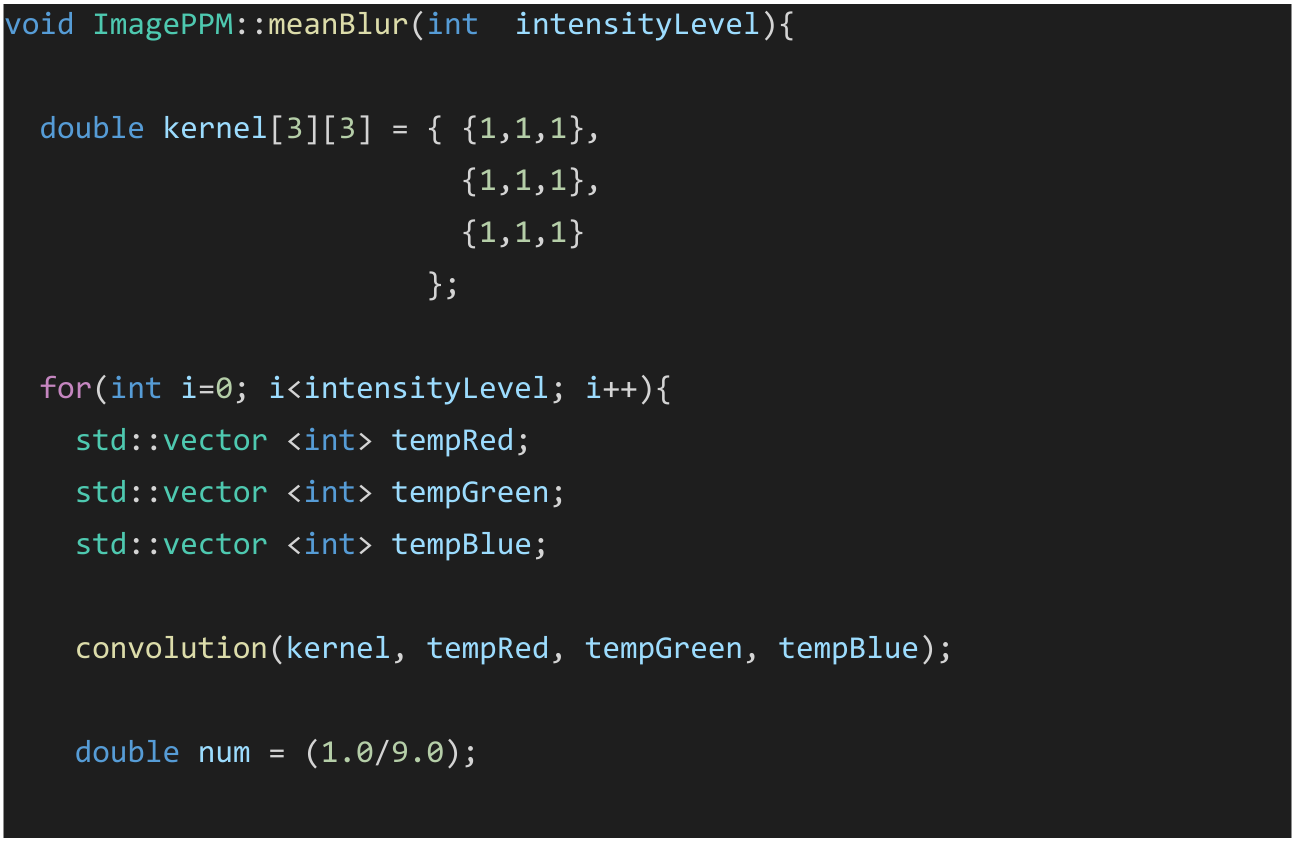
```
void ImagePPM::sobel(void){

  grayScale();

  double g_x[3][3] = { {0.125, 0, -0.125},
                       {0.25, 0, -0.25},
                       {0.125, 0, -0.125}
                    };

  double g_y[3][3] = { {0.125, 0.25, 0.125},
                       {0, 0, 0},
                       {-0.125, -0.25, -0.125}
                    };

  std::vector<int> salida_x;
  std::vector<int> salida_y;

  std::vector<int> v(1, -1);

  // to see why we pass v that way as a parameter, go check the
  // method convolution on MatrixRGB.cpp
  convolution(g_x, salida_x, v, v);
  convolution(g_y, salida_y, v, v);

  std::vector<unsigned int> salida_final;
  salida_final.resize(red.size());

  for(int i=0; i<salida_x.size(); i++){
        salida_final[i]  =  int(  round(  sqrt(  pow(salida_x[i],  2)  +
pow(salida_y[i], 2) ) ) ) ;
  }

  red = salida_final;
  green = salida_final;
  blue = salida_final;
  std::cout<< "salida size: "<<salida_x.size() <<"\n ";

}
```



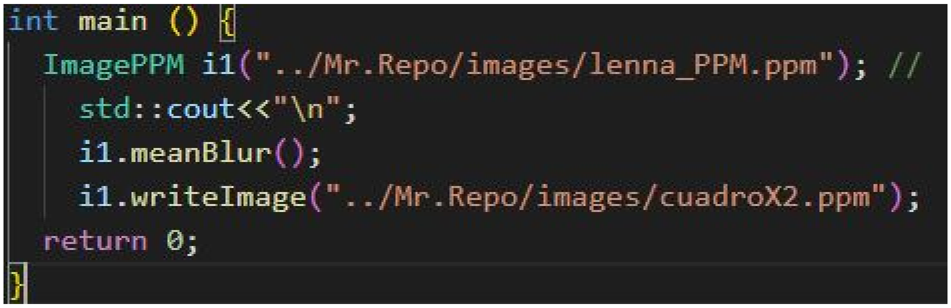
```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.sobel();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```



6. Prueba filtro MeanBlur

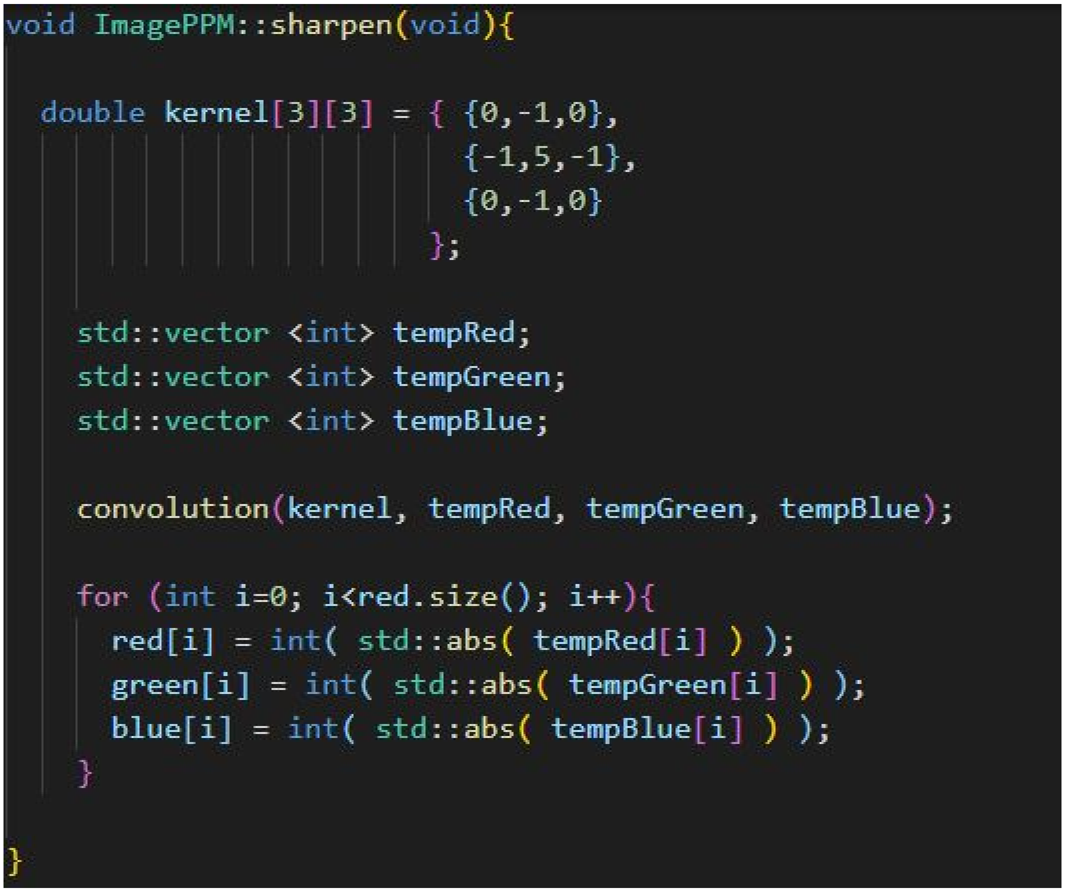


```
void ImagePPM::meanBlur(int  intensityLevel){

  double kernel[3][3] = { {1,1,1},
                          {1,1,1},
                          {1,1,1}
                        };

  for(int i=0; i<intensityLevel; i++){
    std::vector <int> tempRed;
    std::vector <int> tempGreen;
    std::vector <int> tempBlue;

    convolution(kernel, tempRed, tempGreen, tempBlue);

    double num = (1.0/9.0);
    multiplyVectorBy(num, tempRed);
    multiplyVectorBy(num, tempGreen);
    multiplyVectorBy(num, tempBlue);

    for (int i=0; i<red.size(); i++){
      red[i] = tempRed[i];
      green[i] = tempGreen[i];
      blue[i] = tempBlue[i];
    }

  }

}
```

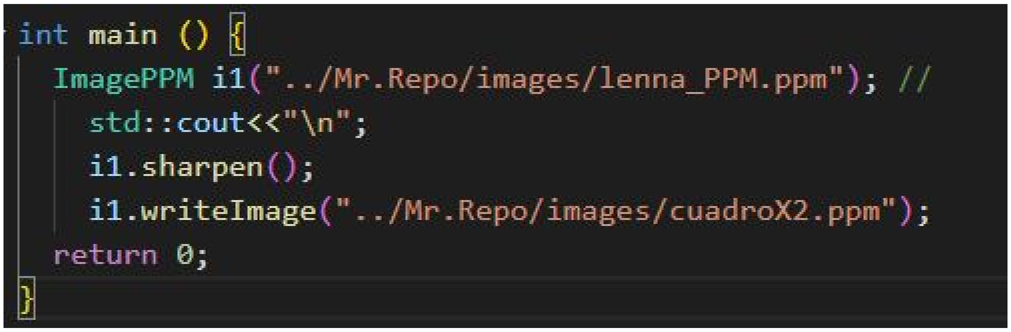


```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.meanBlur();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

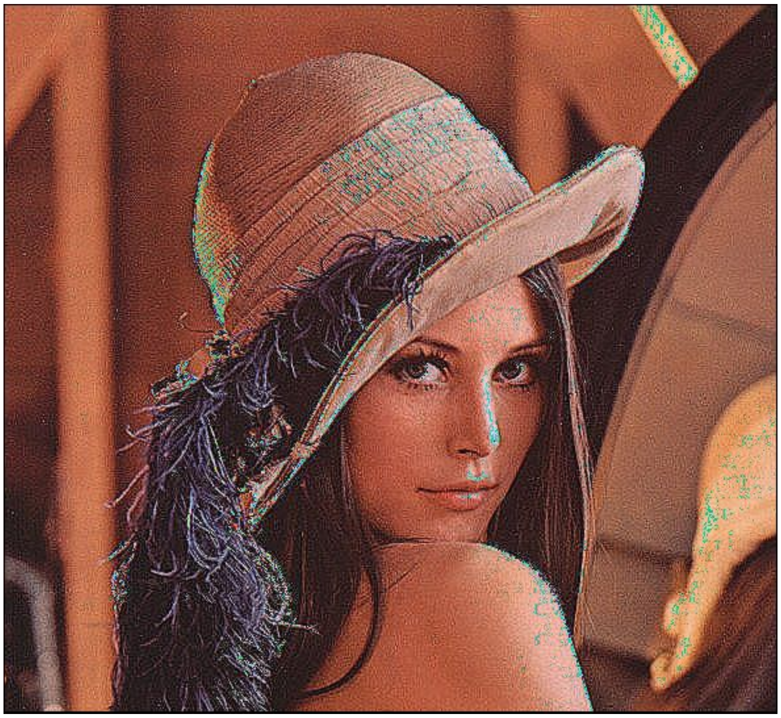
7. Prueba filtro Sharpen



```
void ImagePPM::sharpen(void){

  double kernel[3][3] = { {0,-1,0},
                          {-1,5,-1},
                          {0,-1,0}
                        };

    std::vector <int> tempRed;
    std::vector <int> tempGreen;
    std::vector <int> tempBlue;

    convolution(kernel, tempRed, tempGreen, tempBlue);

    for (int i=0; i<red.size(); i++){
      red[i] = int( std::abs( tempRed[i] ) );
      green[i] = int( std::abs( tempGreen[i] ) );
      blue[i] = int( std::abs( tempBlue[i] ) );
    }

}
```

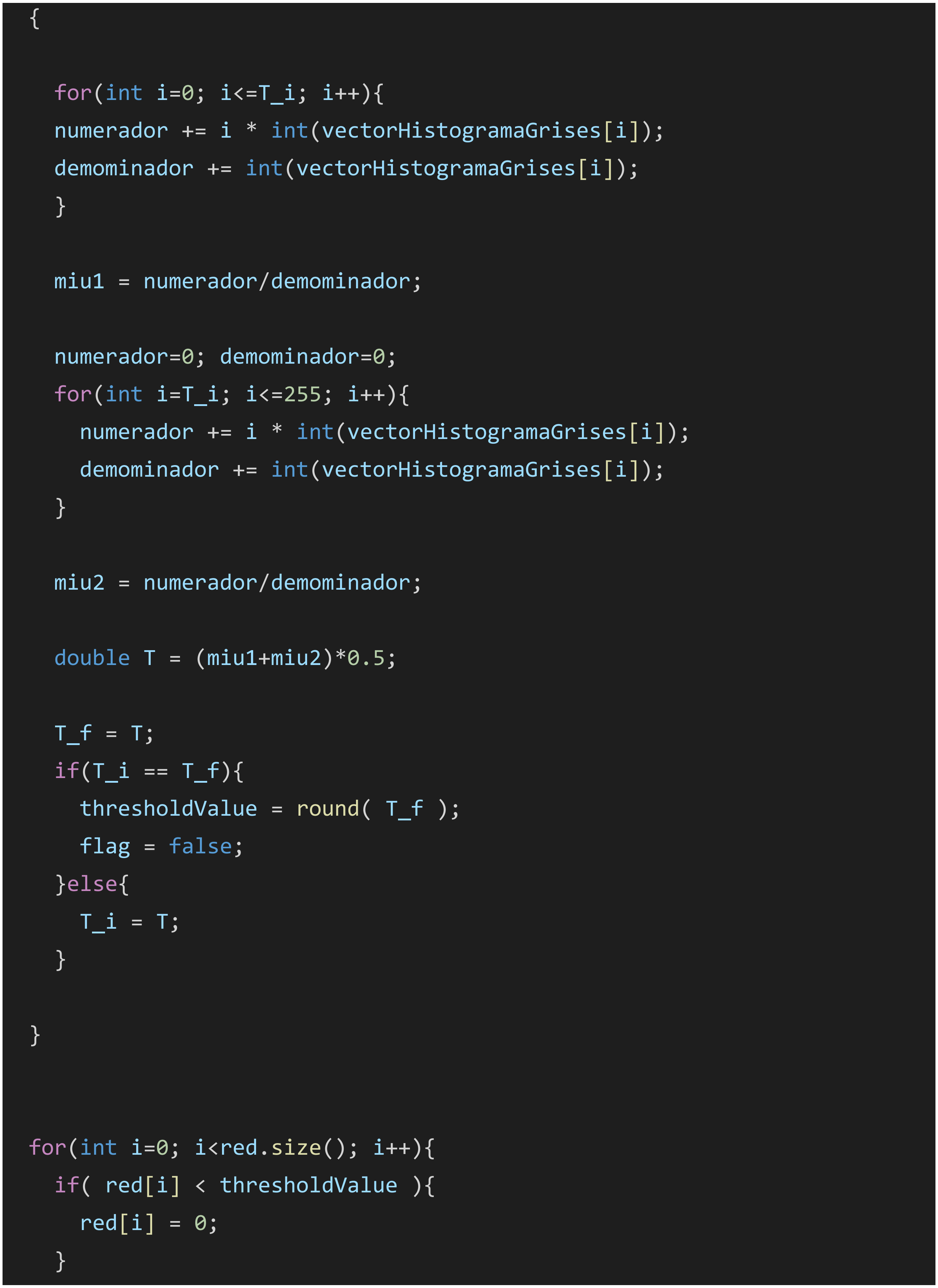


```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.sharpen();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

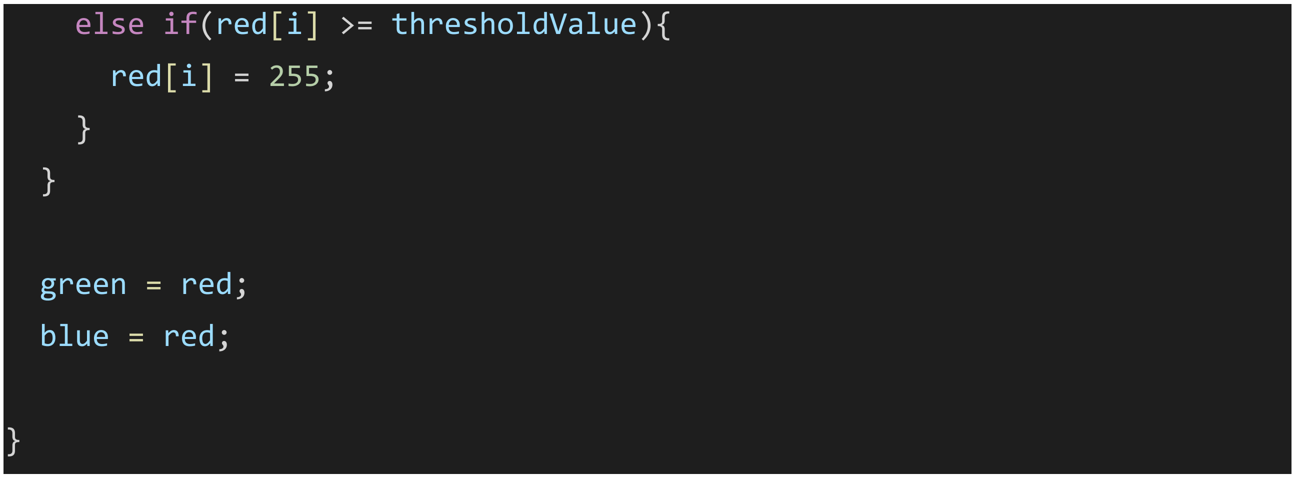


8. ***Prueba filtro threshold***

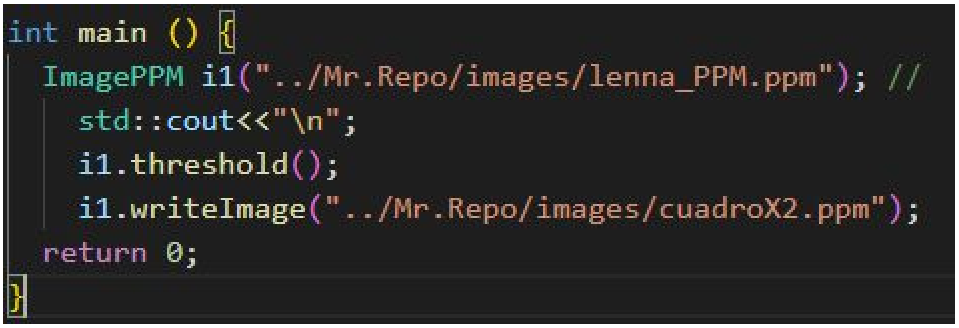
```
void ImagePPM::threshold(void){

  if(vectorHistogramaGrises.empty()){
    grayScale();
    histogramaGrises();
  }

  double miu1, miu2, numerador =0, demominador =0;
  bool flag = true;
  double T_i = 127, T_f = 0, thresholdValue=0;

  while (flag)
```



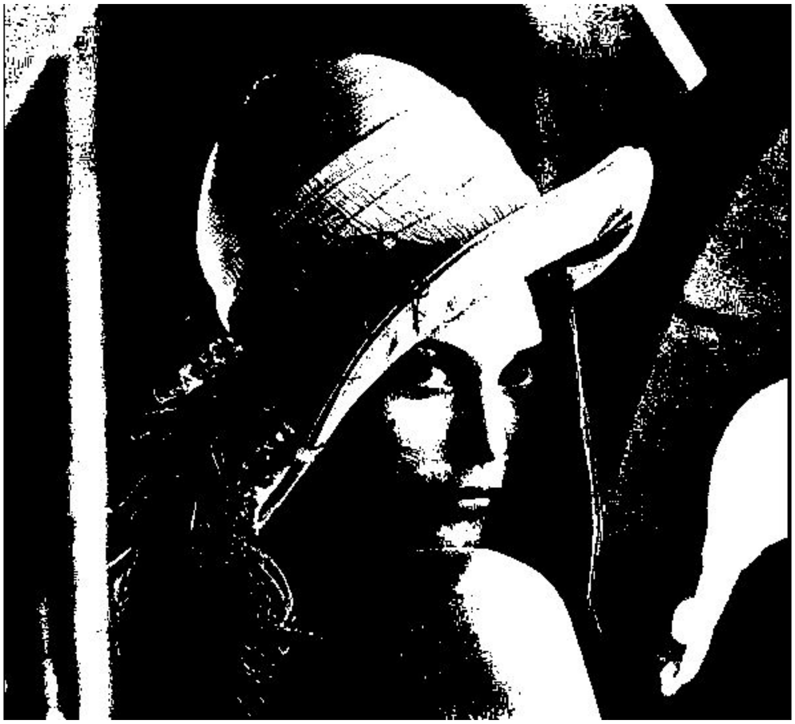
```
{

 for(int i=0; i<=T_i; i++){
 numerador += i * int(vectorHistogramaGrises[i]);
 demominador += int(vectorHistogramaGrises[i]);
 }

 miu1 = numerador/demominador;

 numerador=0; demominador=0;
 for(int i=T_i; i<=255; i++){
  numerador += i * int(vectorHistogramaGrises[i]);
  demominador += int(vectorHistogramaGrises[i]);
 }

 miu2 = numerador/demominador;

 double T = (miu1+miu2)*0.5;

 T_f = T;
 if(T_i == T_f){
  thresholdValue = round( T_f );
  flag = false;
 }else{
  T_i = T;
 }

}



for(int i=0; i<red.size(); i++){
 if( red[i] < thresholdValue ){
  red[i] = 0;
 }
```



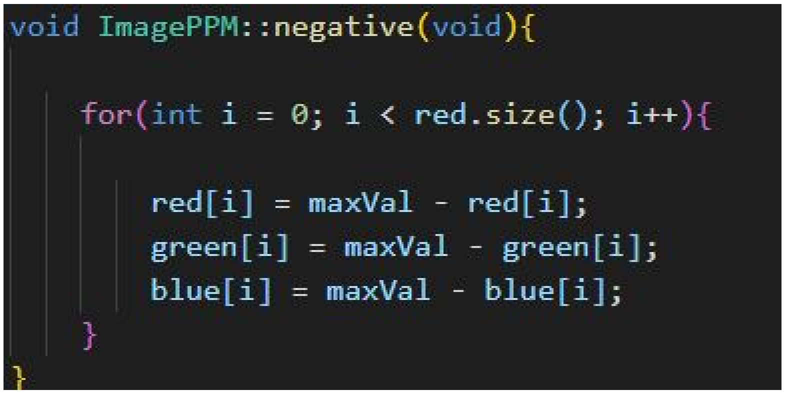
```
    else if(red[i] >= thresholdValue){
      red[i] = 255;
    }
  }

  green = red;
  blue = red;

}
```



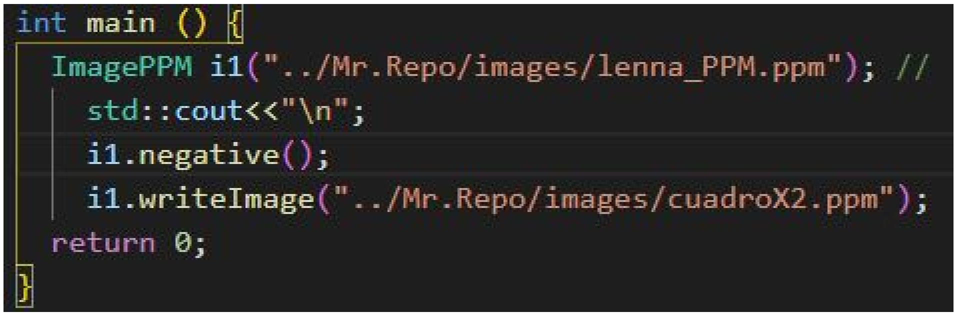
```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.threshold();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```



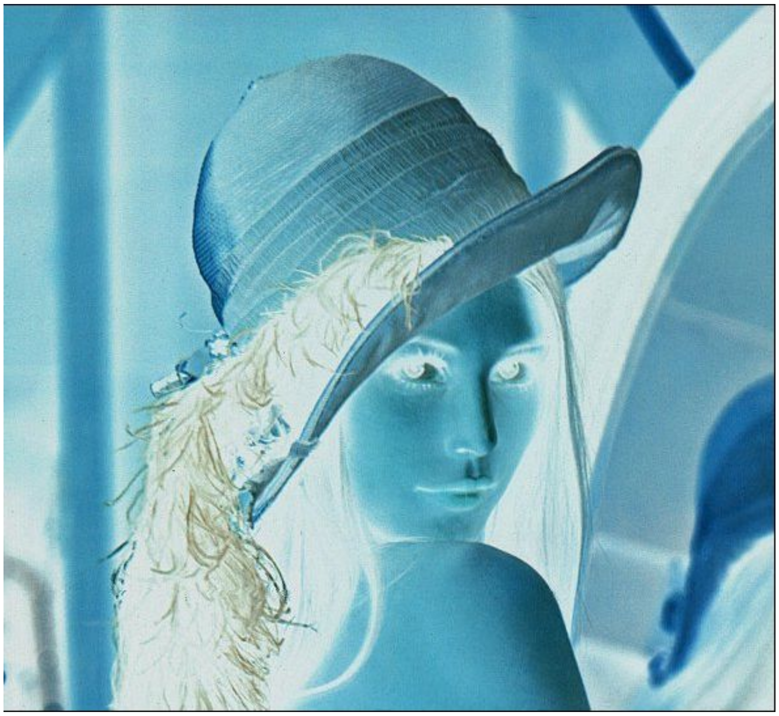
9. Prueba filtro negative



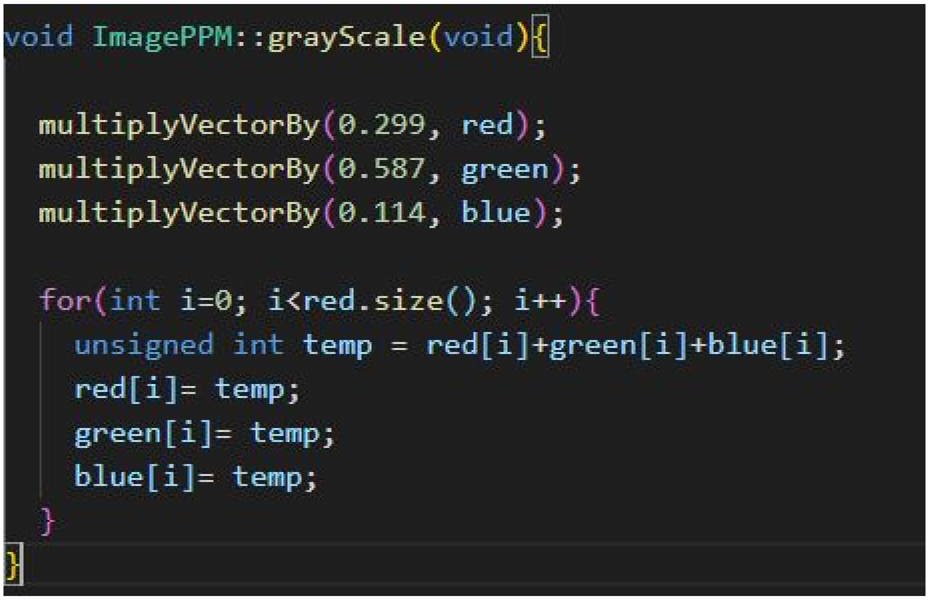
```
void ImagePPM::negative(void){

    for(int i = 0; i < red.size(); i++){

        red[i] = maxVal - red[i];
        green[i] = maxVal - green[i];
        blue[i] = maxVal - blue[i];
    }
}
```



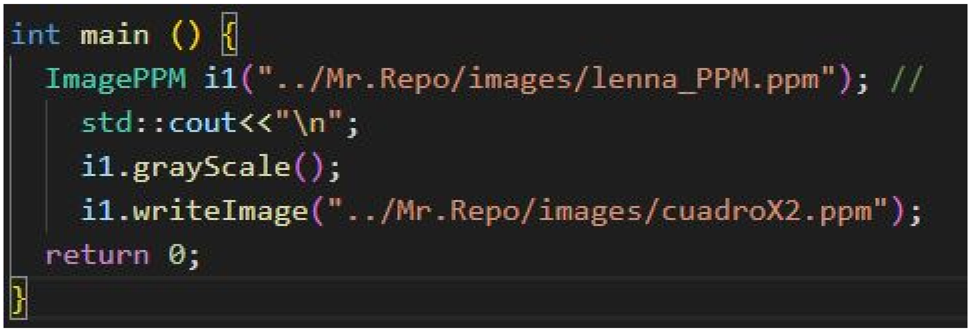
```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.negative();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```



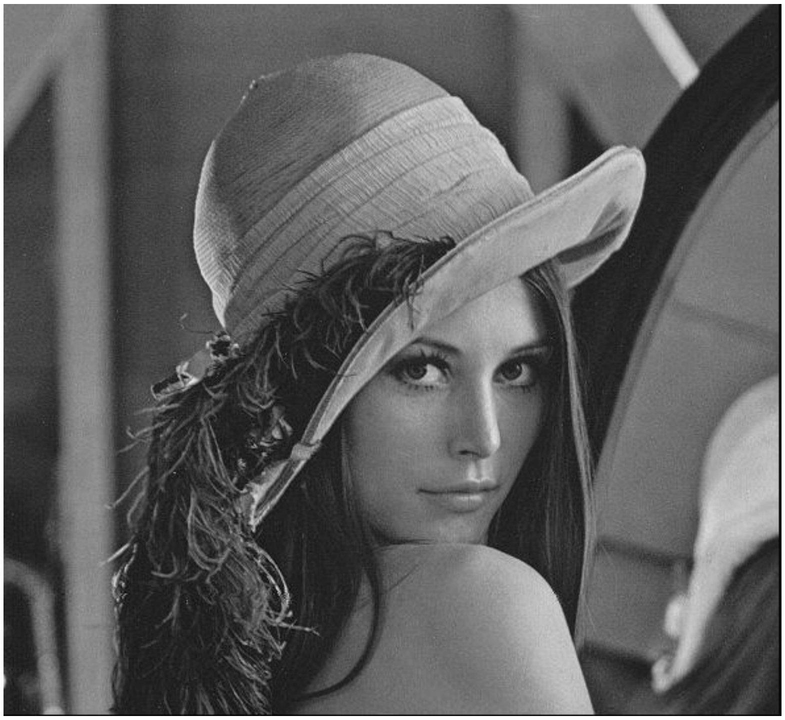
10. Prueba filtro gray scale



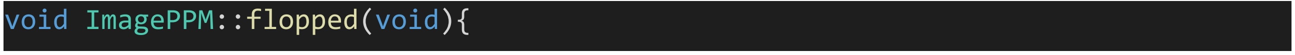
```
void ImagePPM::grayScale(void){

  multiplyVectorBy(0.299, red);
  multiplyVectorBy(0.587, green);
  multiplyVectorBy(0.114, blue);

  for(int i=0; i<red.size(); i++){
    unsigned int temp = red[i]+green[i]+blue[i];
    red[i]= temp;
    green[i]= temp;
    blue[i]= temp;
  }
}
```



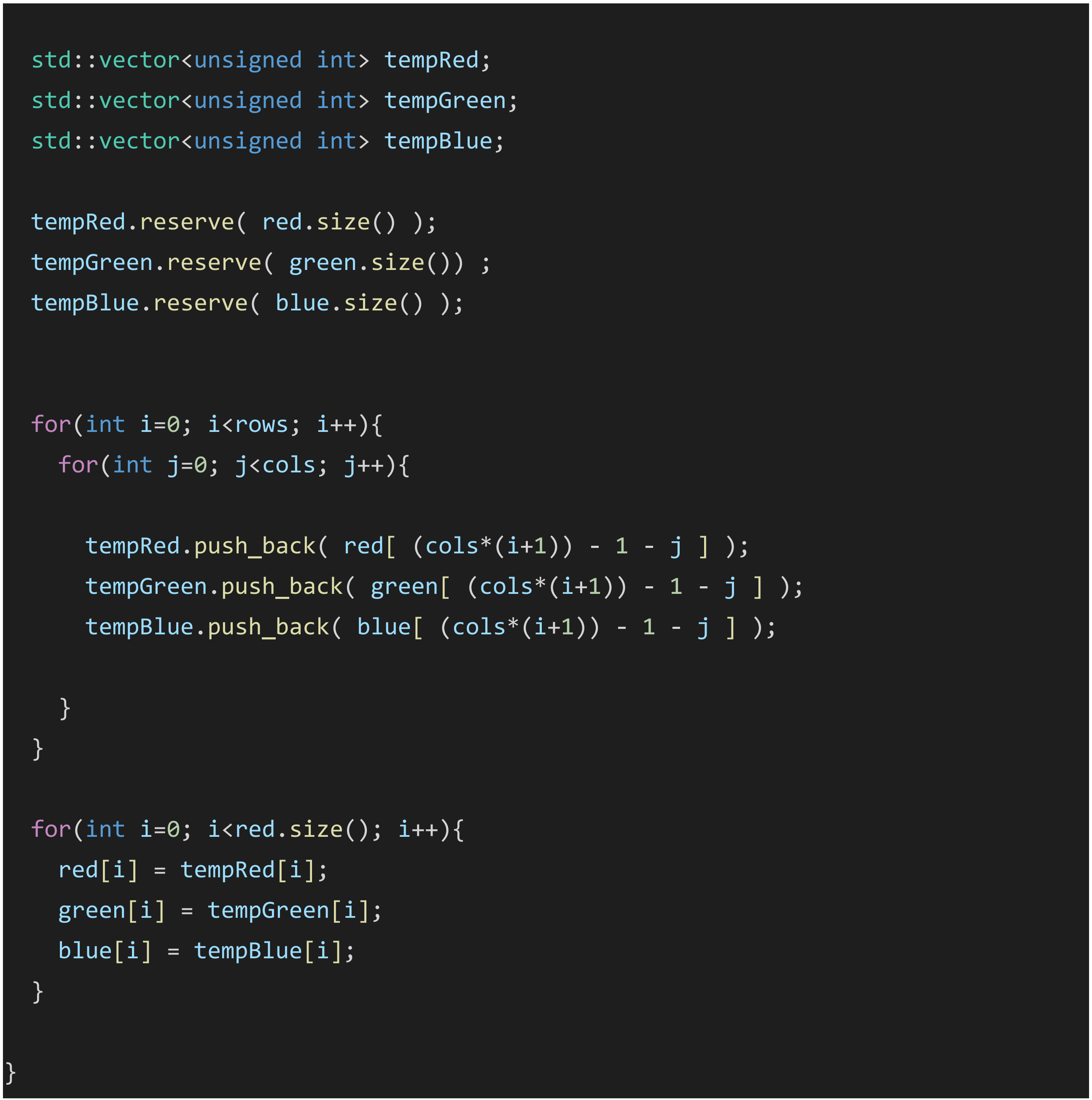
```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.grayScale();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```



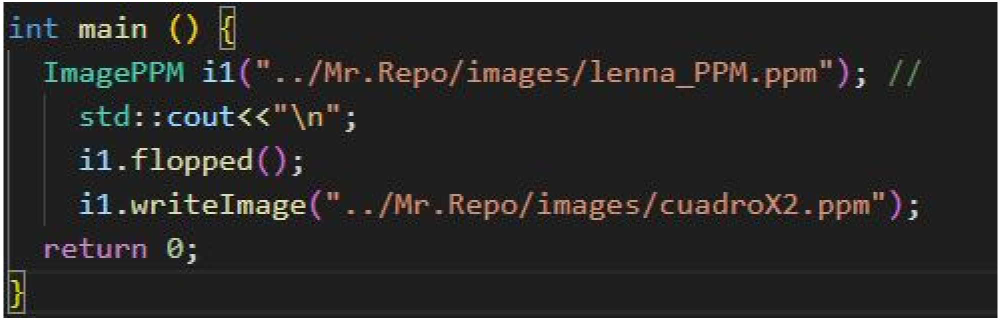
11. Prueba filtro flopped



```
void ImagePPM::flopped(void){
```

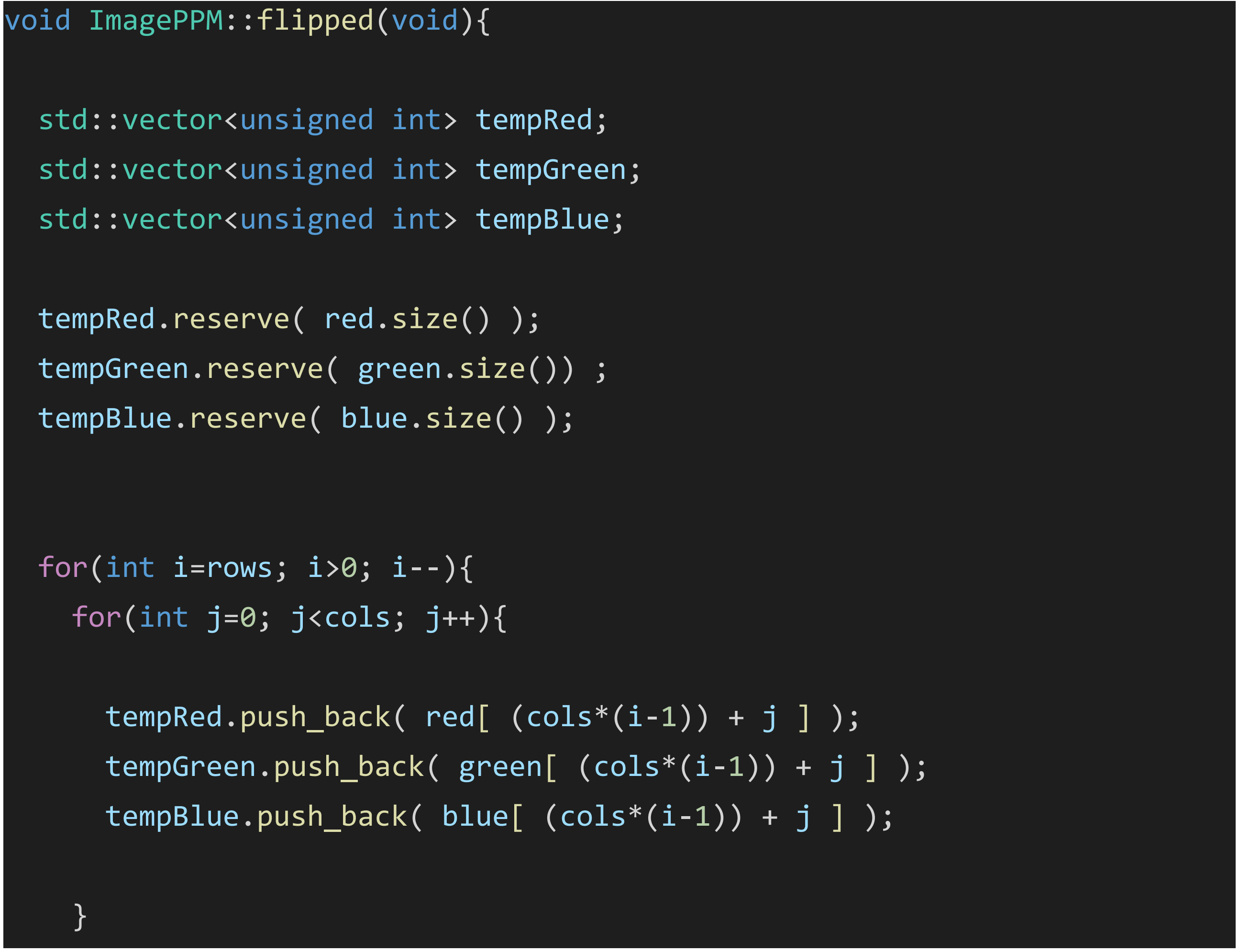


```
  std::vector<unsigned int> tempRed;
  std::vector<unsigned int> tempGreen;
  std::vector<unsigned int> tempBlue;

  tempRed.reserve( red.size() );
  tempGreen.reserve( green.size()) ;
  tempBlue.reserve( blue.size() );


  for(int i=0; i<rows; i++){
    for(int j=0; j<cols; j++){

      tempRed.push_back( red[ (cols*(i+1)) - 1 - j ] );
      tempGreen.push_back( green[ (cols*(i+1)) - 1 - j ] );
      tempBlue.push_back( blue[ (cols*(i+1)) - 1 - j ] );

    }
  }

  for(int i=0; i<red.size(); i++){
    red[i] = tempRed[i];
    green[i] = tempGreen[i];
    blue[i] = tempBlue[i];
  }


}
```

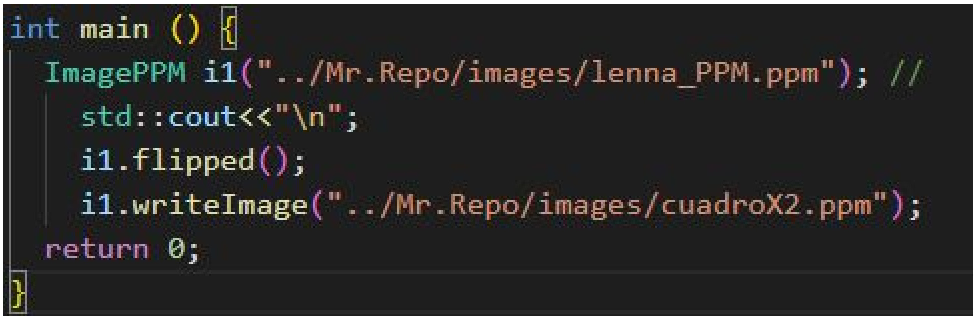


```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.flopped();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

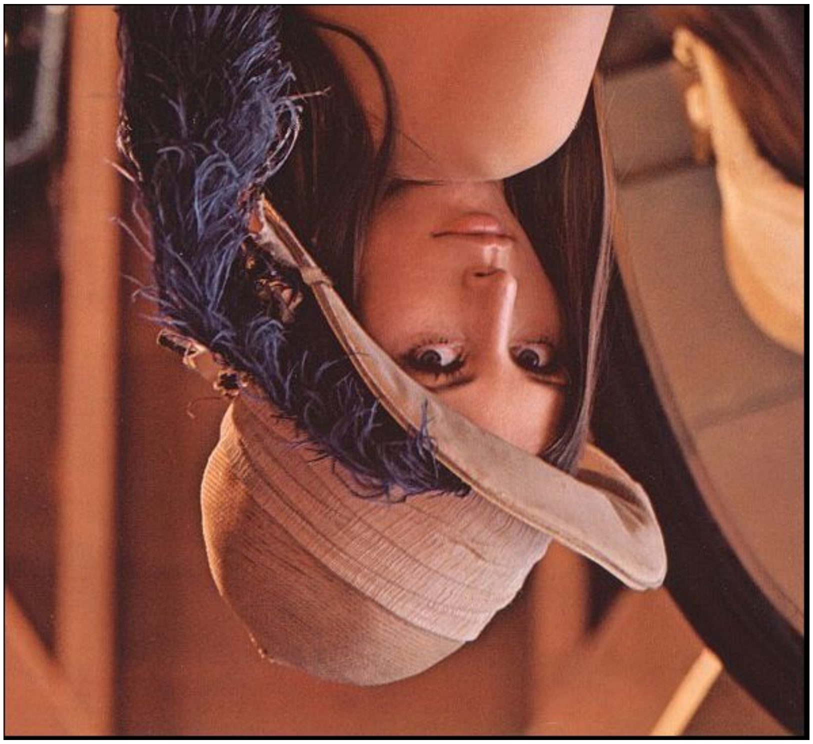
12. Prueba filtro flipped



```
void ImagePPM::flipped(void){

  std::vector<unsigned int> tempRed;
  std::vector<unsigned int> tempGreen;
  std::vector<unsigned int> tempBlue;

  tempRed.reserve( red.size() );
  tempGreen.reserve( green.size()) ;
  tempBlue.reserve( blue.size() );



  for(int i=rows; i>0; i--){
    for(int j=0; j<cols; j++){

      tempRed.push_back( red[ (cols*(i-1)) + j ] );
      tempGreen.push_back( green[ (cols*(i-1)) + j ] );
      tempBlue.push_back( blue[ (cols*(i-1)) + j ] );

    }
  }

  for(int i=0; i<red.size(); i++){
    red[i] = tempRed[i];
    green[i] = tempGreen[i];
    blue[i] = tempBlue[i];
  }

}
```

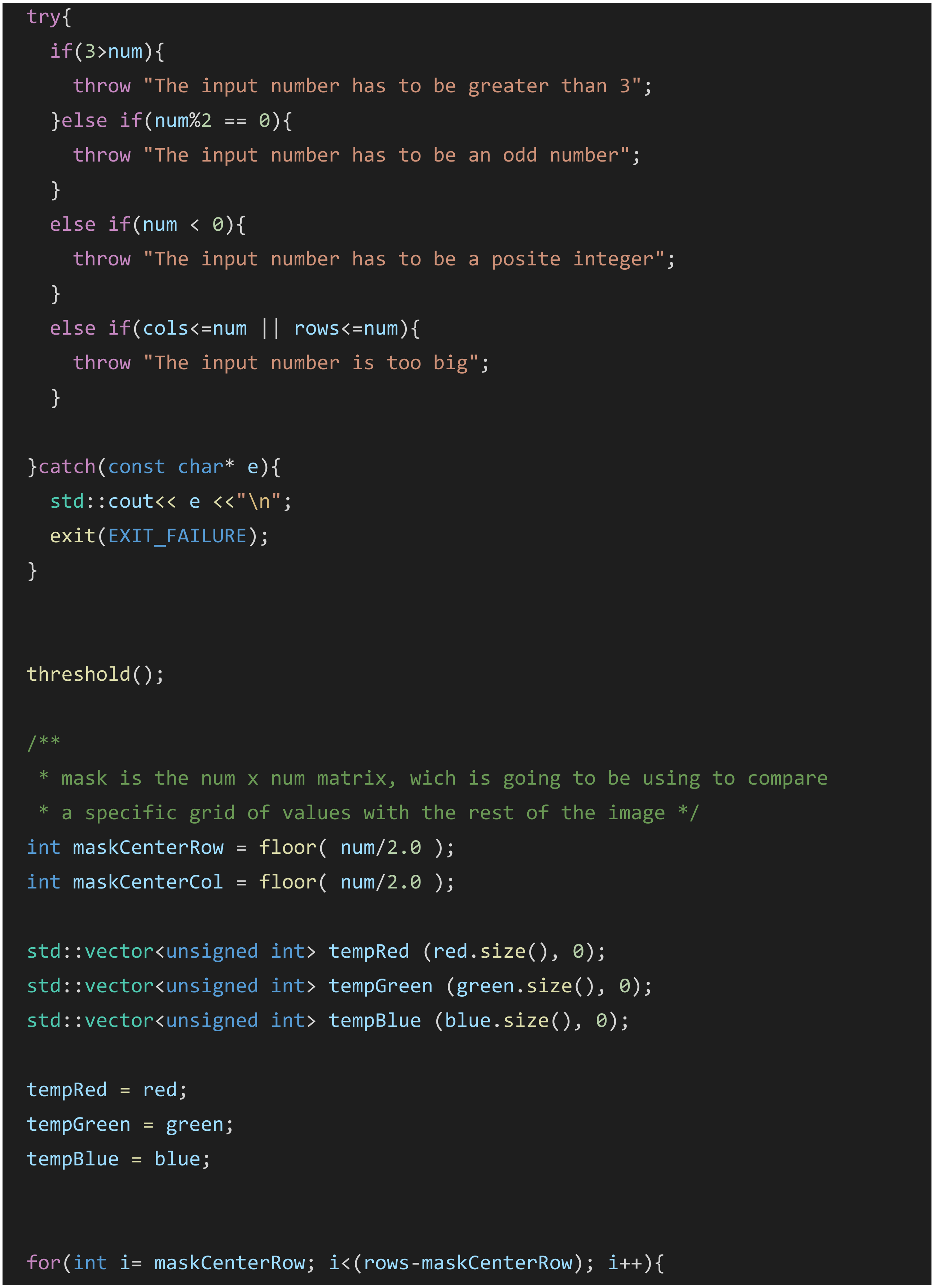


```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.flipped();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

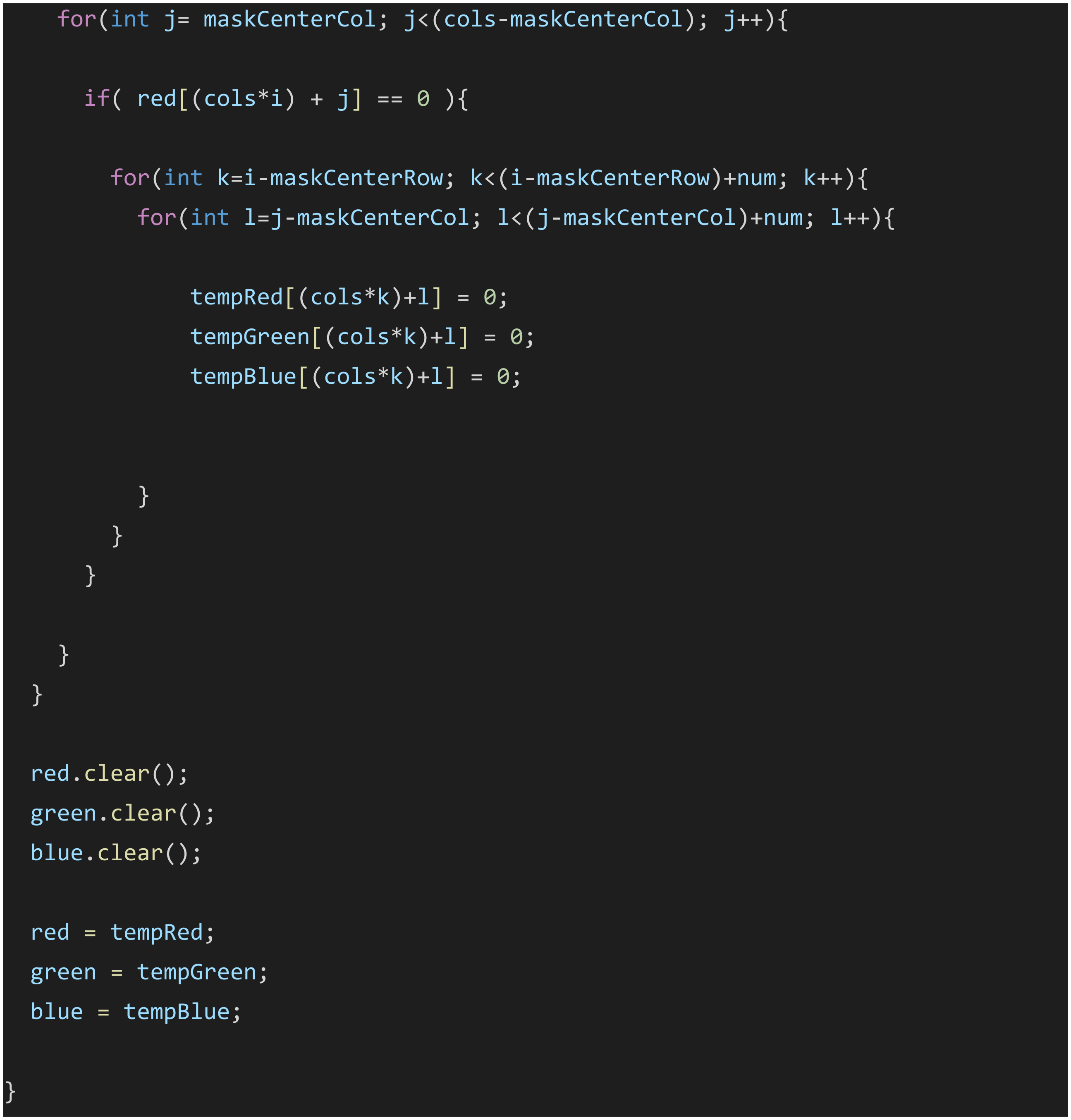


13. Prueba filtro erosion

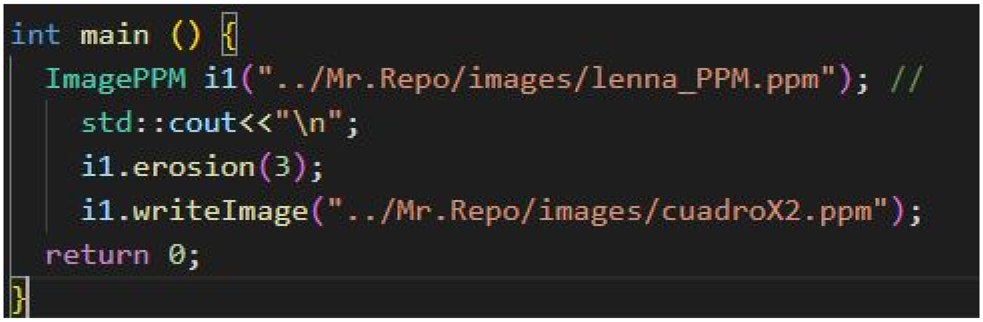
```
void ImagePPM::erosion(int num){
```



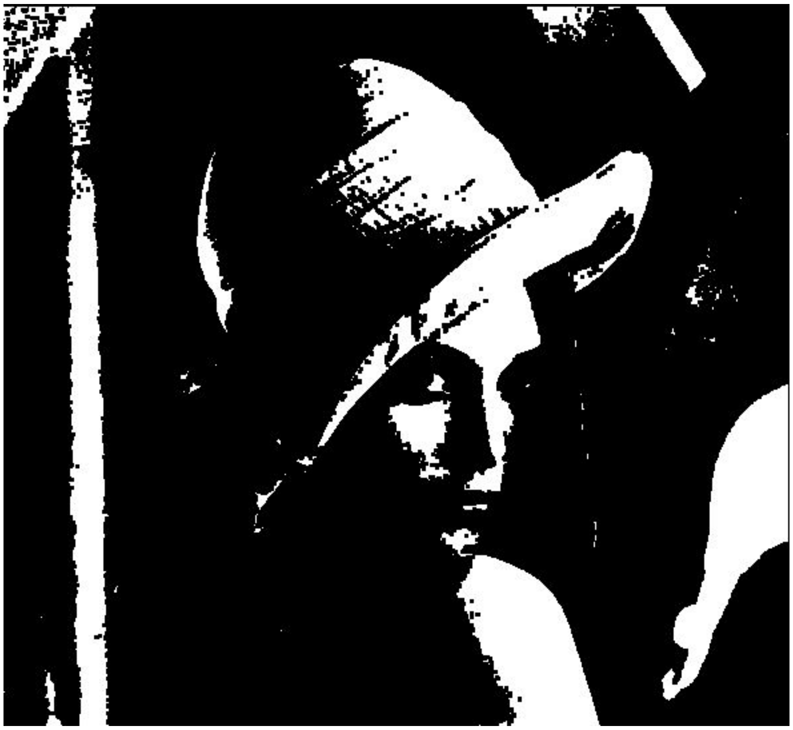
```
try{
  if(3>num){
    throw "The input number has to be greater than 3";
  }else if(num%2 == 0){
    throw "The input number has to be an odd number";
  }
  else if(num < 0){
    throw "The input number has to be a posite integer";
  }
  else if(cols<=num || rows<=num){
    throw "The input number is too big";
  }

}catch(const char* e){
  std::cout<< e <<"\n";
  exit(EXIT_FAILURE);
}



threshold();

/**
 * mask is the num x num matrix, wich is going to be using to compare
 * a specific grid of values with the rest of the image */
int maskCenterRow = floor( num/2.0 );
int maskCenterCol = floor( num/2.0 );

std::vector<unsigned int> tempRed (red.size(), 0);
std::vector<unsigned int> tempGreen (green.size(), 0);
std::vector<unsigned int> tempBlue (blue.size(), 0);

tempRed = red;
tempGreen = green;
tempBlue = blue;



for(int i= maskCenterRow; i<(rows-maskCenterRow); i++){
```



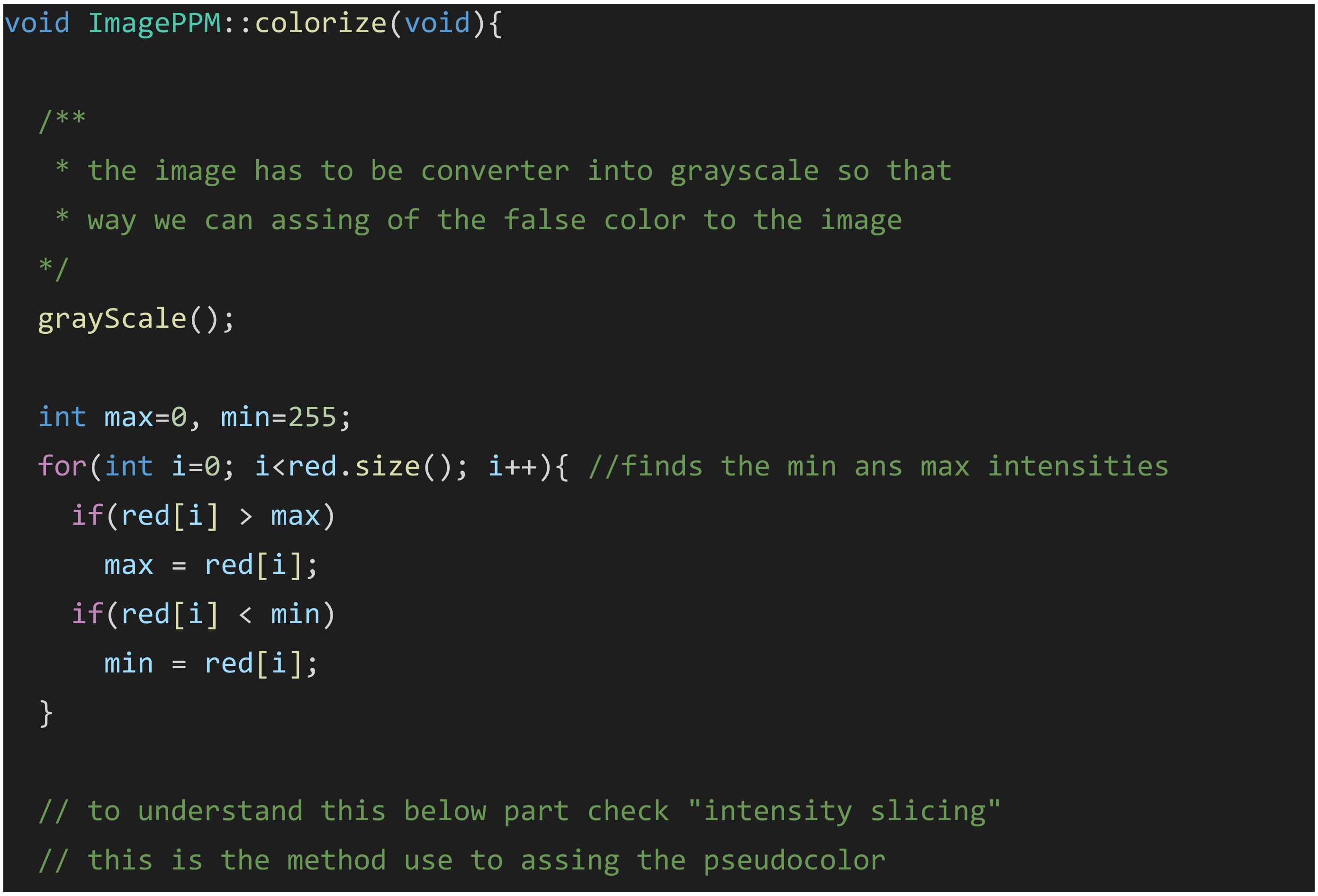
```
    for(int j= maskCenterCol; j<(cols-maskCenterCol); j++){

      if( red[(cols*i) + j] == 0 ){

        for(int k=i-maskCenterRow; k<(i-maskCenterRow)+num; k++){
          for(int l=j-maskCenterCol; l<(j-maskCenterCol)+num; l++){

              tempRed[(cols*k)+l] = 0;
              tempGreen[(cols*k)+l] = 0;
              tempBlue[(cols*k)+l] = 0;


          }
        }
      }

    }
  }

  red.clear();
  green.clear();
  blue.clear();

  red = tempRed;
  green = tempGreen;
  blue = tempBlue;

}
```



```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.erosion(3);
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

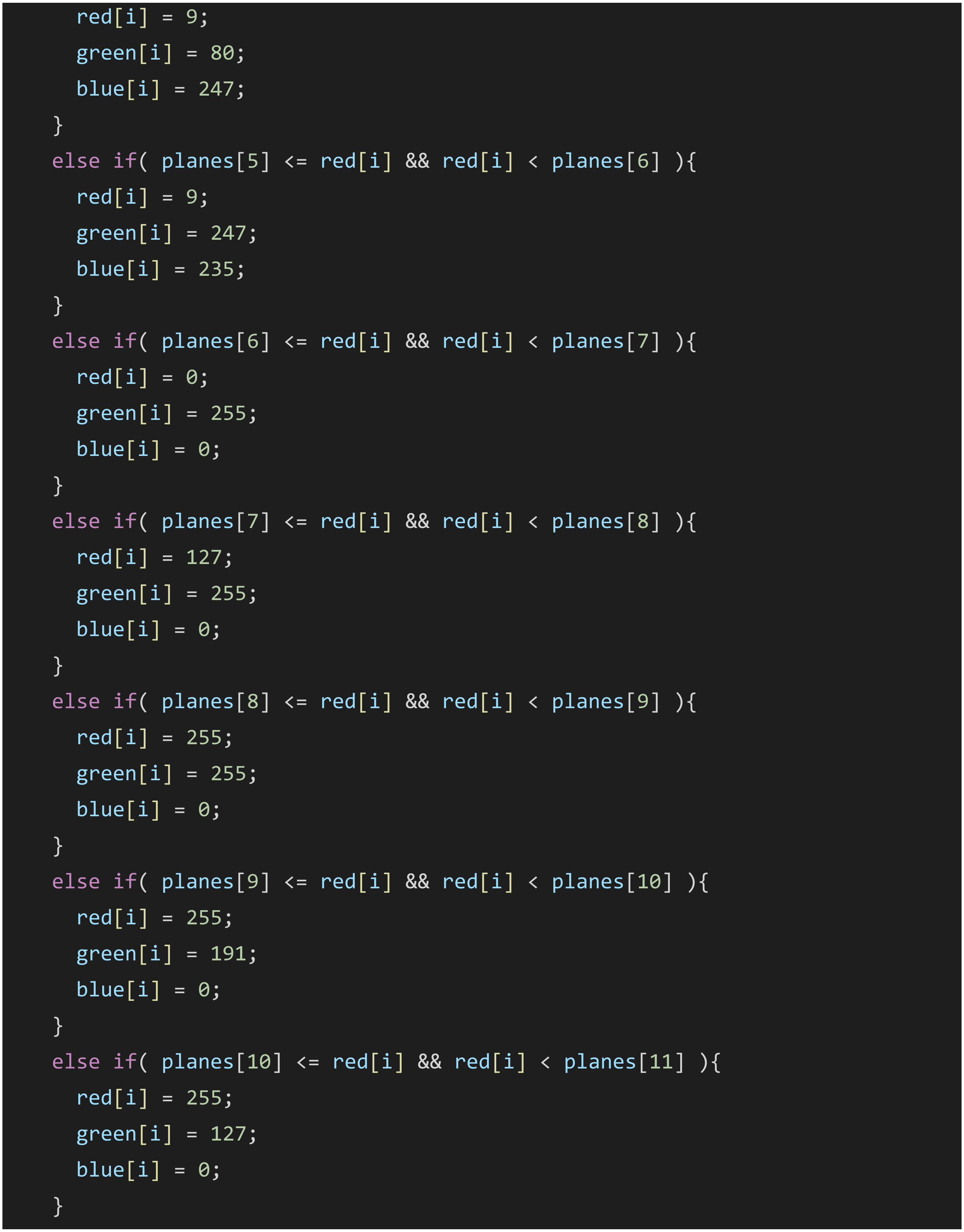


14. Prueba filtro Colorize

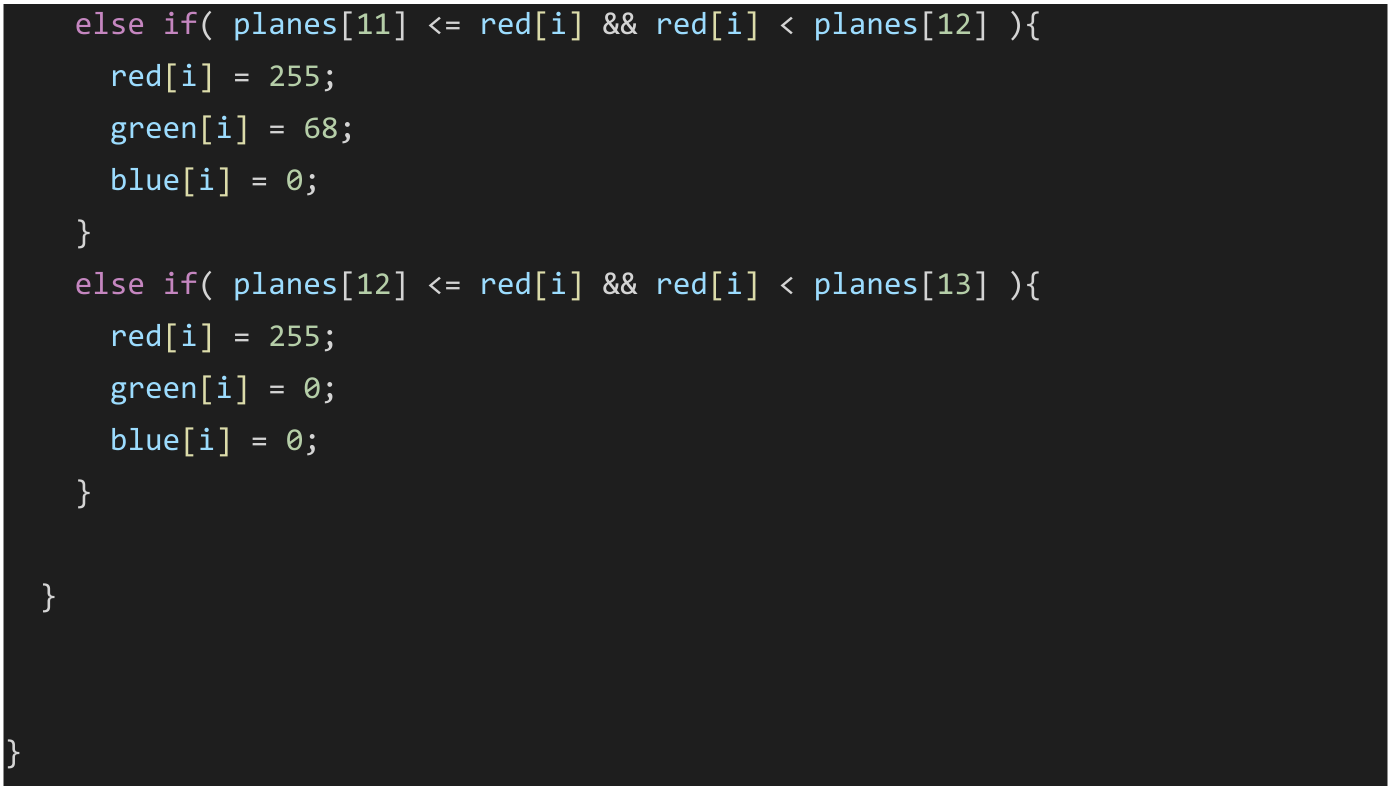


```
void ImagePPM::colorize(void){

  /**
   * the image has to be converter into grayscale so that
   * way we can assing of the false color to the image
  */
  grayScale();

  int max=0, min=255;
  for(int i=0; i<red.size(); i++){ //finds the min ans max intensities
    if(red[i] > max)
      max = red[i];
    if(red[i] < min)
      min = red[i];
  }

  // to understand this below part check "intensity slicing"
  // this is the method use to assing the pseudocolor
```

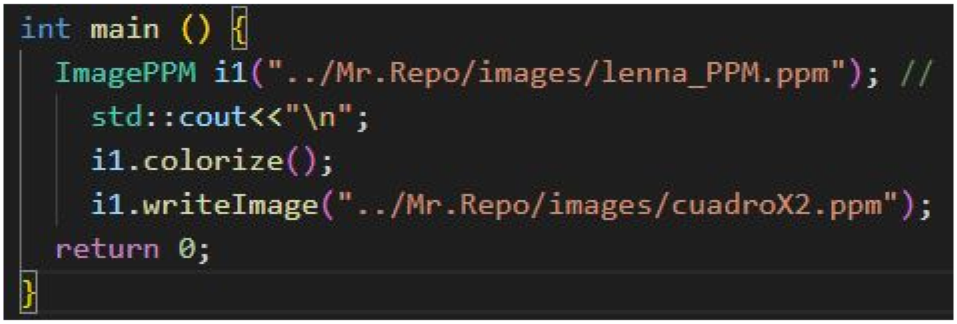
```
int planeDistance, planes[14];
planeDistance = round( (max-min)/13.0 );


planes[0]=min;
planes[14-1]=max;
for(int i=1; i<13; i++){
  planes[i] = planes[i-1] + planeDistance;
}


for(int i=0; i<red.size(); i++){

  if( planes[0] <= red[i] && red[i] < planes[1] ){
    red[i] = 148;
    green[i] = 0;
    blue[i] = 211;
  }
  else if( planes[1] <= red[i] && red[i] < planes[2] ){
    red[i] = 75;
    green[i] = 0;
    blue[i] = 130;
  }
  else if( planes[2] <= red[i] && red[i] < planes[3] ){
    red[i] = 38;
    green[i] = 0;
    blue[i] = 193;
  }
  else if( planes[3] <= red[i] && red[i] < planes[4] ){
    red[i] = 0;
    green[i] = 0;
    blue[i] = 255;
  }
  else if( planes[4] <= red[i] && red[i] < planes[5] ){
```



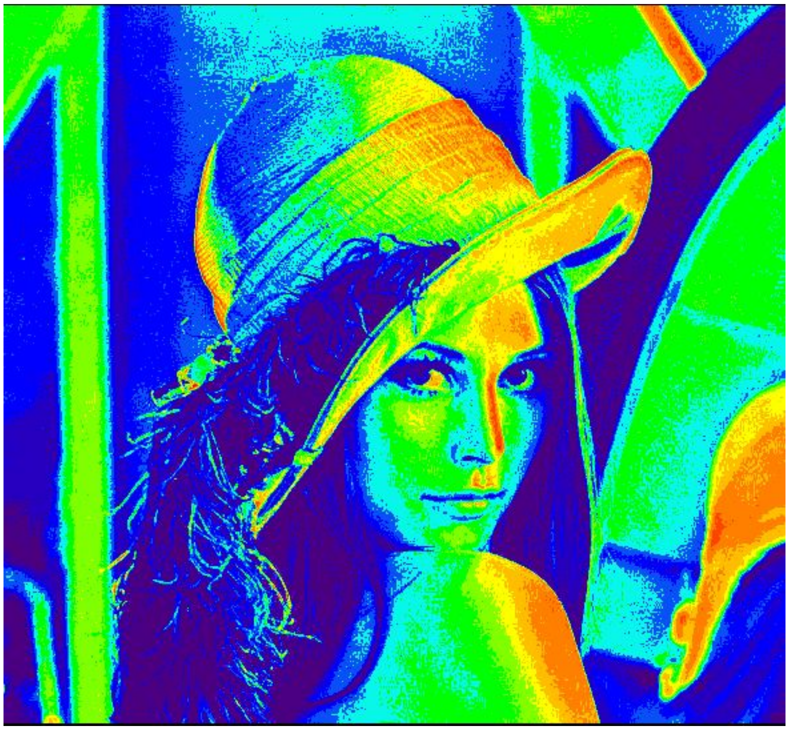
```
      red[i] = 9;
      green[i] = 80;
      blue[i] = 247;
    }
    else if( planes[5] <= red[i] && red[i] < planes[6] ){
      red[i] = 9;
      green[i] = 247;
      blue[i] = 235;
    }
    else if( planes[6] <= red[i] && red[i] < planes[7] ){
      red[i] = 0;
      green[i] = 255;
      blue[i] = 0;
    }
    else if( planes[7] <= red[i] && red[i] < planes[8] ){
      red[i] = 127;
      green[i] = 255;
      blue[i] = 0;
    }
    else if( planes[8] <= red[i] && red[i] < planes[9] ){
      red[i] = 255;
      green[i] = 255;
      blue[i] = 0;
    }
    else if( planes[9] <= red[i] && red[i] < planes[10] ){
      red[i] = 255;
      green[i] = 191;
      blue[i] = 0;
    }
    else if( planes[10] <= red[i] && red[i] < planes[11] ){
      red[i] = 255;
      green[i] = 127;
      blue[i] = 0;
    }
```



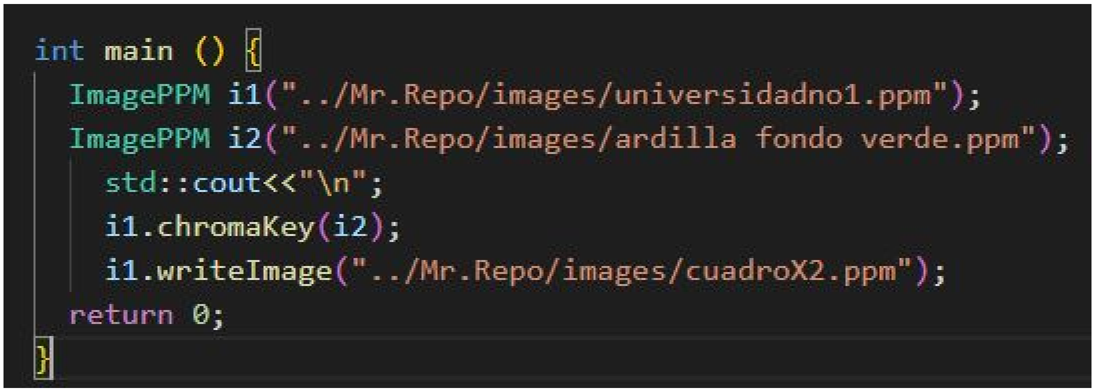
```
    else if( planes[11] <= red[i] && red[i] < planes[12] ){
      red[i] = 255;
      green[i] = 68;
      blue[i] = 0;
    }
    else if( planes[12] <= red[i] && red[i] < planes[13] ){
      red[i] = 255;
      green[i] = 0;
      blue[i] = 0;
    }

  }


}
```



```
int main () {
  ImagePPM i1("../Mr.Repo/images/lenna_PPM.ppm"); //
    std::cout<<"\n";
    i1.colorize();
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

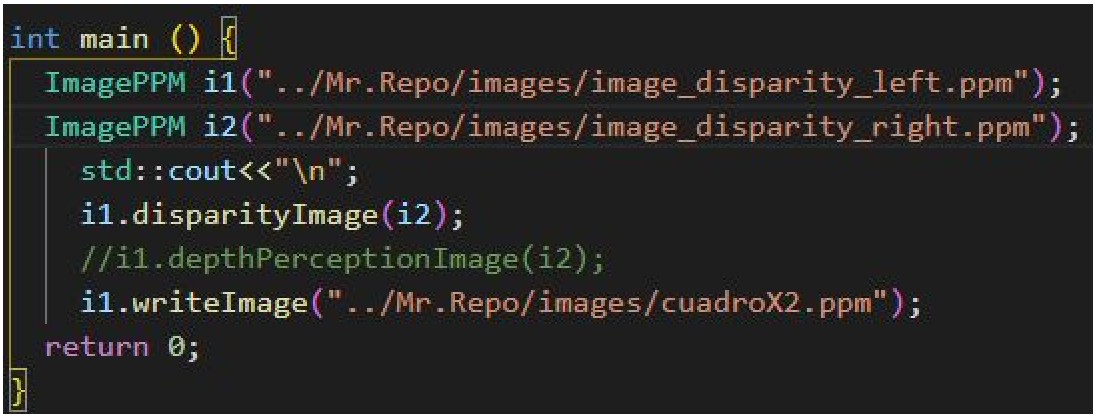


15. Prueba Combinación

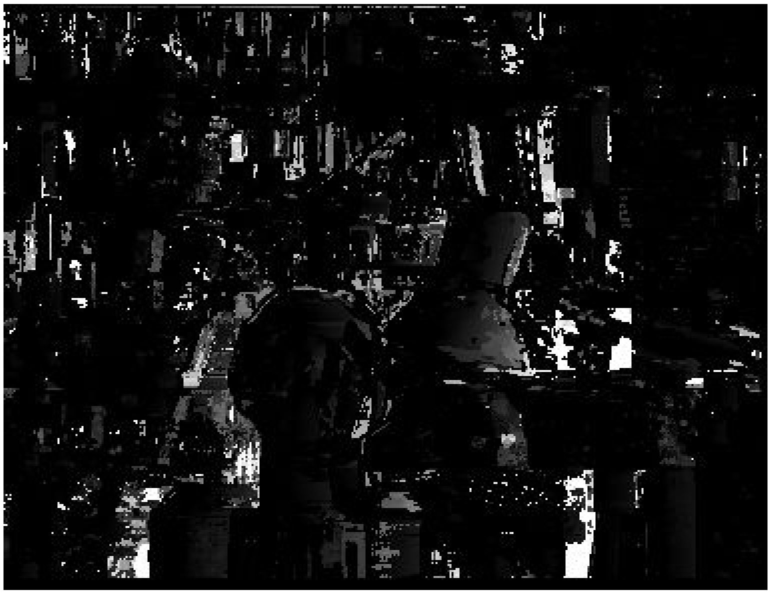


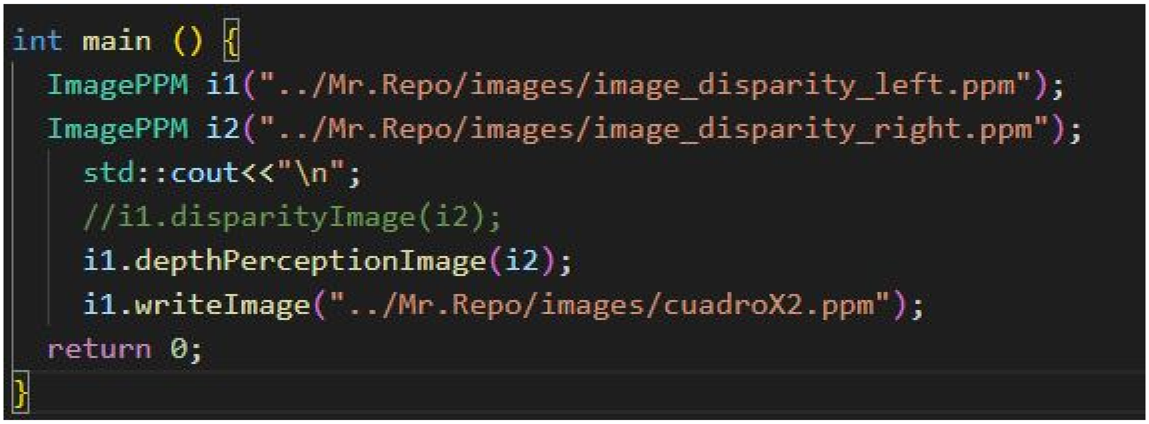
```
int main () {
  ImagePPM i1("../Mr.Repo/images/universidadno1.ppm");
  ImagePPM i2("../Mr.Repo/images/ardilla fondo verde.ppm");
    std::cout<<"\n";
    i1.chromaKey(i2);
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```

prueba disparidad



```
int main () {
  ImagePPM i1("../Mr.Repo/images/image_disparity_left.ppm");
  ImagePPM i2("../Mr.Repo/images/image_disparity_right.ppm");
    std::cout<<"\n";
    i1.disparityImage(i2);
    //i1.depthPerceptionImage(i2);
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```





```
int main () {
  ImagePPM i1("../Mr.Repo/images/image_disparity_left.ppm");
  ImagePPM i2("../Mr.Repo/images/image_disparity_right.ppm");
    std::cout<<"\n";
    //i1.disparityImage(i2);
    i1.depthPerceptionImage(i2);
    i1.writeImage("../Mr.Repo/images/cuadroX2.ppm");
  return 0;
}
```