```
#include <iostream>
#include <vector>
#include <exception>
#include <fstream>
#include <sstream>
#include "GrayLevelImage2D.hpp"
using namespace std;

// pas besoin d'ecrire les constructeurs par copie et d'affectation car ils sont generes automatiquement

// mise en place du type
typedef unsigned char GrayLevel; // le type pour les niveaux de gris.

// declare method of class GrayLevelImage2d.hpp
GrayLevelImage2D::GrayLevelImage2D()
{
    m_width = 0;
    m_height = 0;
}

GrayLevelImage2D::GrayLevelImage2D(int w, int h, GrayLevel g)
{
    m_width = w;
    m_height = h;
    // resize rempoli automatiquement de 0
    m_data.resize(w * h, g);
```

```
}

void GrayLevelImage2D::fill(GrayLevel g)
{
    m_data.assign(m_width * m_height, g);
}

int GrayLevelImage2D::w() const
{
    return m_width;
}

int GrayLevelImage2D::h() const
{
    return m_height;
}

GrayLevel GrayLevelImage2D::at(int i, int j) const
{
    return m_data[i + j * m_width];
}

GrayLevel &GrayLevelImage2D::at(int i, int j)
{
    return m_data[i + j * m_width];
}
```

```
// index function
int GrayLevelImage2D::index(int i, int j) const
{
    return i + j * m_width;
}

GrayLevelImage2D::Iterator::Iterator(GrayLevelImage2D &image, int x, int y)
    : Container::iterator(image.m_data.begin() + image.index(x, y))
{
}

GrayLevelImage2D::Iterator GrayLevelImage2D::begin()
{
    return Iterator(*this, 0, 0);
}

GrayLevelImage2D::Iterator GrayLevelImage2D::end()
{
    return Iterator(*this, 0, m_height);
    // ou return Iterator(*this, m_width, m_height-1);
}

GrayLevelImage2D::Iterator GrayLevelImage2D::start(int x, int y)
{
    return Iterator(*this, x, y);
```

```
}

std::pair<int, int> GrayLevelImage2D::position(Iterator it) const
{
    int x = it - m_data.begin();
    int y = x / m_width;
    x = x % m_width;
    return std::make_pair(x, y);
}

string readline(std::istream &input)
{
    string str;
    do
    {
        getline(input, str);
    } while (str != "" && str[0] == '#');
    return str;
}

bool GrayLevelImage2D::importPGM(std::istream &input)
{
    if (!input.good())
        return false;
    std::string format = readline(input);
```

```
std::string line = readline(input);
std::string delim = " ";
m_width = std::stoi(line.substr(0, line.find(delim)));
line.erase(0, line.find(delim) + delim.length());
m_height = std::stoi(line);
std::cout << m_width << " " << m_height << " " << format << std::endl;
std::cout << readline(input) << std::endl; // grayscale range
fill(0);
if (format == "P5")
{
    input >> std::noskipws;
    unsigned char v;
    for (Iterator it = begin(), itE = end(); it != itE; ++it)
    {
        input >> v;
        *it = v;
    }
}
else
{
    input >> std::skipws;
    int v;
    for (Iterator it = begin(), itE = end(); it != itE; ++it)
    {
        input >> v;
        *it = v;
```

```
        }
    }
    return true;
}

bool GrayLevelImage2D::exportPGM(ostream &output, bool ascii)
{
    // write header
    output << "P5" << endl;
    output << m_width << " " << m_height << endl;
    output << "255" << endl;
    // write data
    if (ascii)
    {
        for (int j = 0; j < m_height; ++j)
        {
            for (int i = 0; i < m_width; ++i)
            {
                output << (int)at(i, j) << " ";
            }
            output << endl;
        }
    }
    else
    {
        for (Iterator it = begin(), itE = end(); it != itE; ++it)
```

```
        {
            output << *it;
        }
    }
    return true;
}

void GrayLevelImage2D::medianFilter(int k)
{
    GrayLevelImage2D copy(*this);
    for (int j = 0; j < m_height; ++j)
    {
        for (int i = 0; i < m_width; ++i)
        {
            std::vector<GrayLevel> values;
            for (int y = -k; y <= k; ++y)
            {
                for (int x = -k; x <= k; ++x)
                {
                    if (i + x >= 0 && i + x < m_width && j + y >= 0 && j + y < m_height)
                    {
                        values.push_back(copy.at(i + x, j + y));
                    }
                }
            }
            std::sort(values.begin(), values.end());
```

```
            at(i, j) = values[values.size() / 2];
        }
    }
}

void GrayLevelImage2D::convolation(double coefficient)
{
    GrayLevelImage2D copy(*this);
    for (int j = 0; j < m_width; ++j)
    {
        for (int i = 0; i < m_height; ++i)
        {

            //le if permettent de verifier si on est sur la bordure de l'image ou pas
            double newVal = at(i, j) * (1 + coefficient);
            if (i > 0)
                newVal -= at(i - 1, j) * (coefficient / 4);
            if (i < m_width - 1)
                newVal -= at(i + 1, j) * (coefficient / 4);
            if (j > 0)
                newVal -= at(i, j - 1) * (coefficient / 4);
            if (j < m_height - 1)
                newVal -= at(i, j + 1) * (coefficient / 4);
            copy.at(i, j) = newVal;
        }
    }
```

```
    *this = copy;

}
```

```
#ifndef _GRAYLEVELIMAGE2D_HPP_
#define _GRAYLEVELIMAGE2D_HPP_

#include <iostream>
#include <vector>
#include <exception>

class GrayLevelImage2D
{

public:
  typedef unsigned char GrayLevel;          // le type pour les niveaux de gris.
  typedef std::vector<GrayLevel> Container; // le type pour stocker les niveaux de gris de l'image.

  /**
     Représente un itérateur sur toutes les valeurs d'une image.

     Model of DefaultConstructible, CopyConstructible, Assignable,
     RandomAccessIterator. */
  struct Iterator : public Container::iterator
  {
    Iterator(GrayLevelImage2D &Image, int x, int y);
  };

public:
  GrayLevelImage2D();
```

```
GrayLevelImage2D(int w, int h, GrayLevel g = 0);
void fill(GrayLevel g);

//! [gli2d-sec3]
/// @return la largeur de l'image.
int w() const;
/// @return la hauteur de l'image.
int h() const;

/**
   Accesseur read-only à la valeur d'un pixel.
   @return la valeur du pixel(i,j)
*/
GrayLevel at(int i, int j) const;

/**
   Accesseur read-write à la valeur d'un pixel.
   @return une référence à la valeur du pixel(i,j)
*/
GrayLevel &at(int i, int j);
//! [gli2d-sec3]

Iterator begin();
Iterator end();
Iterator start(int x, int y);
```

```
  std::pair<int, int> position(Iterator it) const;

  bool importPGM(std::istream &input);
  bool exportPGM(std::ostream &output, bool ascii = true);
  void medianFilter(int k);
  void convolation(double coefficient);

private:
  // Calcule l'indice dans m_data du pixel (x,y).
  int index(int x, int y) const;
  // Le tableau contenant les valeurs des pixels.
  Container m_data;
  // la largeur
  int m_width;
  // la hauteur
  int m_height;
};

#endif // #ifndef _GRAYLEVELIMAGE2D_HPP_
```

```
#include <iostream>
#include <vector>
#include <exception>
#include <fstream>
#include <sstream>
#include "histogramme.hpp"
using namespace std;



Histogramme::Histogramme()
{
    h.resize(256);
    H.resize(256);
}



// Cette classe aura une méthode void init( GrayLevelImage2D & img ), qui parcourera l'image pour calculer son hist
void Histogramme::init(GrayLevelImage2D& img)
{
    for (int i = 0; i < img.w(); i++)
    {
        for (int j = 0; j < img.h(); j++)
        {
            h[img.at(i,j)]++;
        }
    }
```

```
    H[0] = h[0];
    for (int i = 1; i < 256; i++)
    {
        H[i] = H[i-1] + h[i];
    }
}


int Histogramme::egalisation( int j ) const
{
    return 255 * H[j]/(double) H[255];
}
```

```
#ifndef _HISTOGRAMME_HPP
#define _HISTOGRAMME_HPP

#include <iostream>
#include <vector>
#include <exception>
#include "GrayLevelImage2D.hpp"

using namespace std;

class Histogramme
{
public:
    Histogramme();
    void init( GrayLevelImage2D & img );
    int egalisation( int j ) const;

private:
    //histogramme normal
    vector<double> h;
    //histogramme cumulée
    vector<double> H;
};

#endif // #ifndef _HISTOGRAMME_HPP
```

```
#include <iostream>
#include <fstream>
#include "GrayLevelImage2D.hpp"
using namespace std;

int main(int argc, char **argv)
{
    typedef GrayLevelImage2D::GrayLevel GrayLevel;
    typedef GrayLevelImage2D::Iterator Iterator;
    if (argc < 3)
    {
        std::cerr << "Usage: filtre-median <input.pgm> <output.pgm> <k=1>" << std::endl;
        return 0;
    }
    GrayLevelImage2D img;
    ifstream input(argv[1]); // récupère le 1er argument.
    bool ok = img.importPGM(input);
    if (!ok)
    {
        std::cerr << "Error reading input file." << std::endl;
        return 1;
    }
    input.close();

    int k;
    if (argc == 4)
```

```
    {
        string s = argv[3];
        if (s != "")
            //convert s to int
            k = atoi(s.c_str());
        else
            k = 1;
    }
    else
        k = 1;
    img.medianFilter(k);


    ofstream output(argv[2]); // récupère le 2ème argument.
    ok = img.exportPGM(output, false);
    if (!ok)
    {
        std::cerr << "Error writing output file." << std::endl;
        return 1;
    }
    output.close();
    return 0;
}
```

```cpp
// double-brightness.cpp
#include <iostream>
#include <fstream>
#include "GrayLevelImage2D.hpp"
using namespace std;

int testMedianFilter()
{
  typedef GrayLevelImage2D::GrayLevel GrayLevel;
  typedef GrayLevelImage2D::Iterator Iterator;
  // if (argc < 3)
  // {
  //   std::cerr << "Usage: double-brightness <input.pgm> <output.pgm>" << std::endl;
  //   return 0;
  // }
  GrayLevelImage2D img;
  ifstream input("lenaBruit.pgm"); // récupère le 1er argument.
  bool ok = img.importPGM(input);
  if (!ok)
  {
    std::cerr << "Error reading input file." << std::endl;
    return 1;
  }
  input.close();

  img.medianFilter(10);
```

```
  // for (Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
  // {
  //   *it = (2 * (int)(*it)) % 256;
  // }

  ofstream output("lenaBruitSol.pgm"); // récupère le 2ème argument.
  ok = img.exportPGM(output, false);
  if (!ok)
  {
    std::cerr << "Error writing output file." << std::endl;
    return 1;
  }
  output.close();
  return 0;
}

int testConvolution(double coefficient)
{
  typedef GrayLevelImage2D::GrayLevel GrayLevel;
  typedef GrayLevelImage2D::Iterator Iterator;
  // if (argc < 3)
  // {
  //   std::cerr << "Usage: double-brightness <input.pgm> <output.pgm>" << std::endl;
  //   return 0;
  // }
```

```
GrayLevelImage2D img;
ifstream input("lena.pgm"); // récupère le 1er argument.
bool ok = img.importPGM(input);
if (!ok)
{
  std::cerr << "Error reading input file." << std::endl;
  return 1;
}
input.close();


img.convolation(coefficient);


// for (Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
// {
//   *it = (2 * (int)(*it)) % 256;
// }


ofstream output("lenaConvo.pgm"); // récupère le 2ème argument.
ok = img.exportPGM(output, false);
if (!ok)
{
  std::cerr << "Error writing output file." << std::endl;
  return 1;
}
output.close();
return 0;
```

```
}


int orignalMain(int argc, char **argv)
{
  typedef GrayLevelImage2D::GrayLevel GrayLevel;
  typedef GrayLevelImage2D::Iterator Iterator;
  if (argc < 3)
  {
    std::cerr << "Usage: double-brightness <input.pgm> <output.pgm>" << std::endl;
    return 0;
  }
  GrayLevelImage2D img;
  ifstream input(argv[1]); // récupère le 1er argument.
  bool ok = img.importPGM(input);
  if (!ok)
  {
    std::cerr << "Error reading input file." << std::endl;
    return 1;
  }
  input.close();
  for (Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
  {
    *it = (2 * (int)(*it)) % 256;
  }
  ofstream output(argv[2]); // récupère le 2ème argument.
  ok = img.exportPGM(output, false);
```

```
  if (!ok)
  {
    std::cerr << "Error writing output file." << std::endl;
    return 1;
  }
  output.close();
  return 0;
}

int main(int argc, char **argv)
{
  return orignalMain(argc, argv);
}
```

```
#include <iostream>
#include <fstream>
#include "GrayLevelImage2D.hpp"

using namespace std;

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

  {
    //! [tgli2d-sec3]
    GrayLevelImage2D img(8, 8, 5); // imagette 8x8 remplie de 5
    for (int y = 0; y < img.h(); ++y)
      for (int x = 0; x < img.w(); ++x)
        std::cout << " " << (int)img.at(x, y); // la conversion permet de voir les caractères sous forme d'entiers.
    std::cout << std::endl;
    //! [tgli2d-sec3]
  }

  {
    // a verifier le 4 pcq j'ai pas le meme nombre de 5 qui s'affiche
    GrayLevelImage2D img(8, 8, 5); // imagette 8x8 remplie de 5
    //! [tgli2d-sec4]
    for (GrayLevelImage2D::Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
      std::cout << " " << (int)*it; // la conversion permet de voir les caractères sous forme d'entiers.
                                    //! [tgli2d-sec4]
```

```
}


typedef GrayLevelImage2D::GrayLevel GrayLevel;
typedef GrayLevelImage2D::Iterator Iterator;
GrayLevelImage2D img;
ifstream input(argv[1]);
try
{
  img.importPGM(input);
}
catch (char const *msg)
{
  std::cerr << "Exception: " << msg << std::endl;
}
catch (...)
{
  std::cerr << "Exception." << std::endl;
}
input.close();
for (Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
{
  const GrayLevel g = (13 * ((int)(*it))) % 256;
  *it = g;
}
ofstream output(argv[2]);
img.exportPGM(output, false);
```

```
  output.close();

  std::cout << std::endl;

  return 0;

}
```

```
#include <iostream>
#include <fstream>
#include <vector>
#include <exception>
#include "histogramme.hpp"

int main(int argc, char **argv)
{
    typedef GrayLevelImage2D::GrayLevel GrayLevel;
    typedef GrayLevelImage2D::Iterator Iterator;
    if (argc < 2)
    {
        std::cerr << "Usage: filtre-median <input.pgm>" << std::endl;
        return 0;
    }
    GrayLevelImage2D img;
    ifstream input(argv[1]); // récupère le 1er argument.
    bool ok = img.importPGM(input);
    if (!ok)
    {
        std::cerr << "Error reading input file." << std::endl;
        return 1;
    }
    input.close();

    Histogramme histo;
```

```
    histo.init(img);


    // egalisation de l'image
    for (int i = 0; i < img.w(); i++)
    {
        for (int j = 0; j < img.h(); j++)
        {
            img.at(i, j) = histo.egalisation(img.at(i, j));
        }
    }


    // export image
    ofstream output("lenaHistogramme.pgm"); // récupère le 2ème argument.
    ok = img.exportPGM(output, false);
    if (!ok)
    {
        std::cerr << "Error writing output file." << std::endl;
        return 1;
    }
    output.close();
    return 0;
}
```

```
// bruit-impulsionnel.cpp
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <algorithm>
#include "GrayLevelImage2D.hpp"

using namespace std;

double rand01()
{
  return (double)random() / (double)RAND_MAX;
}

int main(int argc, char **argv)
{
  typedef GrayLevelImage2D::GrayLevel GrayLevel;
  typedef GrayLevelImage2D::Iterator Iterator;
  if (argc < 3)
  {
    std::cerr << "Usage: bruit-impulsionnel <input.pgm> <output.pgm> <prob>" << std::endl;
    return 0;
  }
  GrayLevelImage2D img;
  ifstream input(argv[1]); // récupère le 1er argument.
  bool ok = img.importPGM(input);
```

```
  if (!ok)
  {
    std::cerr << "Error reading input file." << std::endl;
    return 1;
  }
  input.close();

  double prob = (argc > 3) ? atof(argv[3]) : 0.01; // récupère la probabilité de bruit
  for (Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
  {
    if (rand01() < prob)
    { // sature dans un sens (noir=0) ou l'autre (blanc=255)
      *it = (rand01() < 0.5) ? 0 : 255;
    }
  }
  ofstream output(argv[2]); // récupère le 2ème argument.
  ok = img.exportPGM(output, false);
  if (!ok)
  {
    std::cerr << "Error writing output file." << std::endl;
    return 1;
  }
  output.close();
  return 0;
}
```

```
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <algorithm>
#include "GrayLevelImage2D.hpp"
#include <math.h>

using namespace std;

double rand01()
{
  return (double)random() / (double)RAND_MAX;
}

int main(int argc, char **argv)
{
  typedef GrayLevelImage2D::GrayLevel GrayLevel;
  typedef GrayLevelImage2D::Iterator Iterator;
  if (argc < 2)
  {
    std::cerr << "Usage: bruit-gaussian <input.pgm> <prob> <power>" << std::endl;
    return 0;
  }
  GrayLevelImage2D img;
  ifstream input(argv[1]);
  bool ok = img.importPGM(input);
```

```
if (!ok)
{
  std::cerr << "Error reading input file." << std::endl;
  return 1;
}
input.close();

double prob = (argc > 2) ? atof(argv[2]) : 0.01;
double power = (argc > 3) ? atof(argv[3]) : 50;

// double prob = 0.35;
// double power = 50;
for (Iterator it = img.begin(), itE = img.end(); it != itE; ++it)
{
  if (rand01() < prob)
  {
    // formule de Box-Muller
    *it = *it + power * (sqrt(-2 * log(rand01())) * cos(2 * M_PI * rand01()));
  }
}
ofstream output("bruited_" + string(argv[1]));
ok = img.exportPGM(output, false);
if (!ok)
{
  std::cerr << "Error writing output file." << std::endl;
  return 1;
```

```
  }
  output.close();
  return 0;
}
```