```
#ifndef THREAD_H_
#define THREAD_H_

#include <pthread.h>

typedef void* thread_run_data_t;
typedef void* (*thread_run_func_t)(thread_run_data_t run_data);

class Thread{
  pthread_t thread;
  thread_run_func_t run_func;
  thread_run_data_t run_data;
public:
  Thread(thread_run_func_t run_func, thread_run_data_t run_data);
  void destroy();
  void start();
  void join(void** result);
  virtual ~Thread();
  static void* starter(void* args);
};

#endif /*THREAD_H_*/
```

```
#include "Thread.h"
#include <iostream> //cout

void* Thread::starter(void* args){
  Thread* thread = (Thread*)args;
  thread→run_func(thread→run_data);
  return NULL;
}

Thread::Thread(thread_run_func_t run_func, thread_run_data_t run_data) {
  this→run_func = run_func;
  this→run_data = run_data;
}

void Thread::destroy(){
}

void Thread::start() {
  //this es el dato
  //starter es la funcion
  pthread_create(&this→thread, NULL, starter, this);
}

void Thread::join(void** result){
  pthread_join(this→thread,result);
}

Thread::~Thread(){}
```

sep 20, 16 17:36 **Position.h** Page 1/1

```
#ifndef POSITION_H_
#define POSITION_H_

#include <iostream> //cout


class Position{
  int column,row;
public:
  Position(int rowNew,int columNew);
  Position relativityPosition(Position& position);
  Position sum(Position& position);
  int getColumn();
  int getRow();
  void setColumn(int columnNew);
  void setRow(int rowNew);
  void print();
  ~Position();
};
#endif /* POSITION_H_ */
```

sep 20, 16 17:36 **Position.cpp** Page 1/1

```
#include "Position.h"



Position::Position(int rowNew,int columNew){
    this→row = rowNew;
    this→column = columNew;
}


  //MÃ©todo que te devuelve posicion relativa.
  Position Position::relativityPosition(Position& otherPosition){
    int columnNew = otherPosition.getColumn() − this→column;
    int rowNew = otherPosition.getRow() − this→row;
    Position newPosition(rowNew,columnNew);
    return newPosition;
  }

  Position Position::sum(Position& position){
    int newColumn = this→column + position.getColumn();
    int newRow = this→row + position.getRow();
    Position newPosition(newRow,newColumn);
    return newPosition;
  }

  void Position::setColumn(int columnNew){
    this→column = columnNew;
  }

  void Position::print(){
    std::cout << "("<< this→row << "," << this→column <<  ")" << std::endl;
  }

  void Position::setRow(int rowNew){
    this→row = rowNew;
  }

  int Position::getColumn(){
    return column;
  }

  int Position::getRow(){
    return row;
  }

  Position::~Position(){}
```

```
#ifndef MATRIX_H_
#define MATRIX_H_

#include "Position.h"
#include <vector>
#include <stdlib.h> // malloc,free
#include <string>

class Matrix{
private:
  bool columnPositionValid(int column);
  bool rowPositionValid(int row);
  int cantRows,cantColumns;
  std::string** matrix; /*puntero a la matriz*/
public:
    Matrix(int rows, int column);
    Matrix(const Matrix& matrix);
    int getCantColumns() const;
    int getCantRows() const;
    std::string getElementPos(int posColumn, int posRows) const;
    std::string getElementPos(Position position) const;
    void setElementPos(int posRows,int posColumn, std::string element);
    bool positionIsValid(Position& position);
    void set(const Matrix& otherMatrix);
    void dimesions();
    void print();
    ~Matrix();
};

#endif /* MATRIX_H_ */
```

```
#include "Matrix.h"
#include <string>

using std::string;
using std::cout;
using std::endl;

Matrix::Matrix(int rows, int column):cantRows(rows),cantColumns(column){
  this→matrix = new string*[cantRows];
  for (int i = 0; i < rows ; i++){
    this→matrix[i]= new string[cantColumns];
  }
}

Matrix::Matrix(const Matrix& otherMatrix):cantRows(otherMatrix.getCantRows()),
cantColumns(otherMatrix.getCantColumns()){
  //cout << "asignation per copy" << endl;
  this→matrix = new string*[cantRows];
  for (int i = 0; i < cantRows ; i++){
    this→matrix[i]= new string[cantColumns];
  }
  int i,j;
  for (i = 1; i ≤ cantRows; i++) {
    for (j = 1; j ≤ cantColumns; j++) {
      this→setElementPos(i,j,otherMatrix.getElementPos(i,j));
    }
  }
}


void Matrix::set(const Matrix& otherMatrix){
  if (otherMatrix.getCantColumns() ≡ this→cantColumns ∧
  otherMatrix.getCantRows() ≡ this→cantRows){
    for (int i = 1; i ≤ cantRows; i++){
      for (int j = 1; j ≤ cantColumns; j++) {
        this→setElementPos(i,j,otherMatrix.getElementPos(i,j));
      }
    }
  }else{
    cout << "no se puede copiar los valores" << endl;
  }
}

//verifica si el num de columna es valido
bool Matrix::columnPositionValid(int column){
    return 0 < column ∧ column ≤ this→cantColumns;
}

//verifica si el num de fila es valido
bool Matrix::rowPositionValid(int row){
    return 0 < row ∧ row ≤ this→cantRows;
  }

//muestra por stdout las dimensiones de la matrix
void Matrix::dimesions(){
  cout << "Tengo col:" <<this→cantColumns<<"y fil:"<<this→cantRows<< endl;
}


void Matrix::print(){
  for (int i = 0; i < this→cantRows; i++) {
    for (int j = 0; j < this→cantColumns; j++) {
      cout << matrix[i][j] << "";
    }
    cout << "" << endl;
  }
  //cout << "++++++++++++++" << endl;
}

//verifica si la posicion fila,colum es valida
bool Matrix::positionIsValid(Position& position){
    return columnPositionValid(position.getColumn()) ^
    rowPositionValid(position.getRow());
  }
  //-------------------------getters---------------------------------
int Matrix::getCantColumns() const {
  return cantColumns;
  }

int Matrix::getCantRows() const{
    return cantRows;
  }

void Matrix::setElementPos(int posRows, int posColumn, string element){
  matrix[posRows-1][posColumn-1] = element;
}

string Matrix::getElementPos(int posRows,int posColumn) const{
  return matrix[posRows-1][posColumn-1];
}

string Matrix::getElementPos(Position position) const{
  return matrix[position.getRow()-1][position.getColumn()-1];
}


Matrix::~Matrix(){
  //cout << "destructor called" << endl;
  for (int i = 0; i < cantRows; i++) {
    delete[] matrix[i];
  }
  delete[](matrix);
}
```

```
#ifndef INTERPRETER_H_
#define INTERPRETER_H_

#include <iostream> //cout
#include <sstream>
#include <string>
#include <vector>

#include "Matrix.h"

class Interpreter{
private:
  void split(const std::string &s, char delim, std::vector<std::string> &elems);
public:
  Interpreter();
  Matrix createMatrix(const std::string& matrix);
  Matrix createMatrix(std::vector<std::string> elems);
  ~Interpreter();
};
#endif /* INTERPRETER_H_ */
```

```cpp
#include "Interpreter.h"
#include <string>
#include <vector>
//Esta se encarga de interpretar una cadena y devolver la matriz
//saca el espacio redundante al principio pero no en el intermedio
using std::string;
using std::stoi;
using std::vector;
using std::stringstream;

Interpreter::Interpreter(){
  //std::cout << "Soy una dilatacion" << std::endl;
}

Interpreter::~Interpreter(){}

Matrix Interpreter::createMatrix(const string& matrix){
  //std::cout << "createMatrix" << std::endl;
  vector<string> elems;
  split(matrix,' ', elems);
  vector<string>::iterator it;
  string::size_type sz;
  it = elems.begin();
  int row = stoi(*it,&sz);
  ++it;
  int colum = stoi(*it,&sz);
  Matrix patron(row,colum);
  //std::cout <<row<<","<<colum<<std::endl;
  int i = 0;
  ++it;
  for (; it≠elems.end() ; ++it) {
    string fila = *it;
    for (int j = 0 ; j < colum ; j++) {
      string elemento = fila.substr(j,1);
      //std::cout << "Elemento:" <<elemento<< std::endl;
      patron.setElementPos(i+1,j+1,elemento);
    }
    i++;
  }
  return patron;
}

Matrix Interpreter::createMatrix(std::vector<string> elems){
  //std::cout << "createMatrix from vector" << std::endl;
  std::string::size_type sz;
  vector<string>::iterator it = elems.begin();
  int row = std::stoi(*it,&sz);
  ++it;
  int colum = std::stoi(*it,&sz);
  Matrix patron(row,colum);
  //std::cout <<row<<","<<colum<<std::endl;
  int i = 0;
  ++it;
  for (; it≠elems.end() ; ++it) {
    string fila = *it;
    for (int j = 0 ; j < colum ; j++) {
      string elemento = fila.substr(j,1);
      //std::cout << "Elemento:" <<elemento<< std::endl;
      patron.setElementPos(i+1,j+1,elemento);
    }
    i++;
  }
  return patron;
}

void Interpreter::split(const string &s, char delim, vector<string> &elems) {
```

```cpp
    stringstream ss;
    ss.str(s);
    string item;
    while (getline(ss, item, delim)) {
        elems.push_back(item);
    }
}
```

```cpp
#include <iostream> //cout
#include <string> //compare
#include <string.h> //compare
#include <vector>  //vector
#include "Position.h"
#include "Matrix.h"
#include "Interpreter.h"
#include "Dilatation.h"
#include "Erosion.h"

using std::string;
using std::cout;
using std::endl;
using std::cin;
using std::vector;

Filter* identifierFilter(string& filterString){
  string dilatationString("d");
  if (¬filterString.compare(dilatationString)){
    Dilatation* dilatation = new Dilatation();
    return dilatation;
  }
  Erosion* erosion = new Erosion();
  return erosion;
}

Matrix getMatrix(char* matrix){
  Interpreter interpreter;
  string matrixString(matrix);
  Matrix oneMatrix = interpreter.createMatrix(matrixString);
  return oneMatrix;
}
//    0      1              2           3         4          5
// ./tp <numero de hilos> <filtro 1> <patron 1> <filtro 2> <patron 2>  ...
int main(int argc, char *argv[]) {
  Interpreter interpreter;
  //int cantThreads = atoi(argv[1]);
  vector<string> vectorImagen;
  //cout << "Cantidad de hilos:" << cantThreads << endl;
  string input_line;
  if (argc < 2){
    cout << "Falta argumentos" << endl;
    return 1;
  }
  if (cin) {
    getline(cin, input_line);
    size_t pos = input_line.find(" ");
    //cout << pos << endl;
    string col =  input_line.substr(0,pos);
    string row = input_line.substr(pos+1);
    vectorImagen.push_back(row);
    vectorImagen.push_back(col);
  }
  while (cin){
    getline(cin, input_line);
    if (input_line.compare("\n") ≡ 1){
      //std::cout << input_line << std::endl;
      vectorImagen.push_back(input_line);
      }
  }
  Matrix matrixOrigin = interpreter.createMatrix(vectorImagen);
  Matrix& image(matrixOrigin);
  //image.print();
  for (int i = 2; i ≤ argc−2; i++) {
    string filterString(argv[i]);
    Filter* filter = identifierFilter(filterString);
```

```cpp
    string matrixString(argv[i+1]);
    Matrix patron = interpreter.createMatrix(matrixString);
    //std::cout << "patron" << endl;
    //patron.print();
    Matrix resultado = filter→aplicateFilter(image,patron);
    image.set(resultado);
    //image.print();
    i++;
  }
  std::cout <<image.getCantColumns()<<" "<<image.getCantRows()<< std::endl;
  image.print();
  return 0;
}
```

```
#ifndef FILTER_H_
#define FILTER_H_

#include "Position.h"
#include "Matrix.h"
#include <list>


class Filter{
private:
  std::list<bool> compareMatrices(Matrix& imagen,Matrix& patron,Position& pos);
  virtual bool checkCoincidence(std::list<bool> lista) = 0;
  Matrix createImageDestin(int row,int column);
public:
    Filter();
    virtual Matrix aplicateFilter(Matrix& image,Matrix& patron);
    ~Filter();
};

#endif /* FILTER_H_ */
```

```
#include "Filter.h"
#include <string>
#include <list>

using std::list;
using std::string;
//Constructor
Filter::Filter(){}

//Destructor
Filter::~Filter(){}

/*Pre:Recibe una matrix imagen y patron y una posicion en la cual se debe
comparar
Post:Devuelve una lista con todos los elementos que se compararon
*/
list<bool> Filter::compareMatrices(Matrix& imagen,Matrix& patron,Position& pos){
  list<bool> lista;
  int row = patron.getCantRows();
  int column = patron.getCantColumns();
  Position posicionMedia(row/2 + 1, column/2 + 1);
  Position posicionRelativa = posicionMedia.relativityPosition(pos);
  Position otherPosition(0,0);
  string asterisco("#");
  bool valor;
  int i,j;
  for (i = 1; i ≤ row; i++) {
    for (j = 1; j ≤ column; j++) {
      otherPosition.setRow(i);
      otherPosition.setColumn(j);
      Position posImagen = posicionRelativa.sum(otherPosition);
      //Posicion no valida
      if (imagen.positionIsValid(posImagen) ≡ 0){
        lista.push_back(false);
      }else{
        //posicion valida
        if (asterisco.compare(patron.getElementPos(i,j)) ≡ 0){
          string elemento = imagen.getElementPos(posImagen);
          valor =  elemento.compare(asterisco);
          if (valor ≡ 0){
            lista.push_back(true);
          }else{
            lista.push_back(false);
          }
        }
      }
    }
  }
    return lista;
  }


Matrix Filter::createImageDestin(int row,int column){
  Matrix destino(row,column);
  for (int i = 1; i ≤ row; i++) {
    for (int j = 1; j ≤ column; j++) {
      destino.setElementPos(i,j,".");
    }
  }
  return destino;
}


Matrix Filter::aplicateFilter(Matrix& image,Matrix& patron){
  Position pivote(0,0);
  int row = image.getCantRows();
  int column = image.getCantColumns();
  Matrix pepe = createImageDestin(row,column);
  for (int i = 1; i ≤ row; i++){
    for (int j = 1; j ≤ column; j++){
      pivote.setRow(i);
      pivote.setColumn(j);
      list<bool> lista = compareMatrices(image,patron,pivote);
      bool valor = checkCoincidence(lista);
      if (valor){
        pepe.setElementPos(pivote.getRow(),pivote.getColumn(),"#");
      }
    }
  }
  //std::cout << "------------------------" << std::endl;
  return pepe;
}
```

```
#ifndef EROSION_H_
#define EROSION_H_
#include "Filter.h"
#include <list>

class Erosion : public Filter{
private:
  bool checkCoincidence(std::list<bool> lista);
public:
  Erosion();
  ~Erosion();
};


#endif /* EROSION_H_ */
```

sep 20, 16 17:36 **Erosion.cpp** Page 1/1

```
#include "Erosion.h"
#include <list>
using std::list;

Erosion::Erosion(){
  //std::cout << "Soy una erosion" << std::endl;
}

Erosion::~Erosion(){}


//Cuando se aplica erosion se necesita chequear si hay coincidencia total
//Pre:Recibe un list de valores booleanos
//Post: Chequea si existe coincidencia total
bool Erosion::checkCoincidence(std::list<bool> lista){
  //std::cout << "checkCoincidence erosion" << std::endl;
  list<bool>::iterator it;
  for (it=lista.begin(); it ≠ lista.end(); ++it){
    bool valor = *it;
    //std::cout << valor << " ";
    if (¬valor){
      return false;
    }
  }
  return true;
}
```

sep 20, 16 17:36 **Dilatation.h** Page 1/1

```
#ifndef DILATATION_H_
#define DILATATION_H_
#include "Filter.h"
#include <list>

class Dilatation : public Filter{
private:
  bool checkCoincidence(std::list<bool> lista);
public:
  Dilatation();
  ~Dilatation();
};


#endif /* DILATATION_H_ */
```

sep 20, 16 17:36 **Dilatation.cpp** Page 1/1

```cpp
#include "Dilatation.h"

#include <list> //list
using std::list;

Dilatation::Dilatation(){}

Dilatation::~Dilatation(){}


//Cuando se aplica erosion se necesita chequear si hay coincidencia total
//Pre:Recibe un list de valores booleanos
//Post: Chequea si existe coincidencia parcial
bool Dilatation::checkCoincidence(std::list<bool> lista){
  //std::cout << "Dilatation checkCoincidence" << std::endl;
  list<bool>::iterator it;
  for (it=lista.begin(); it ≠ lista.end(); ++it){
    bool valor = *it;
    if (valor){
      return true;
    }
  }
  return false;
}
```

sep 20, 16 17:36 **Table of Content** Page 1/1

Table of Contents