

sep 20, 16 17:36

Thread.h

Page 1/1

```

1  #ifndef THREAD_H_
2  #define THREAD_H_
3
4  #include <pthread.h>
5
6  typedef void* thread_run_data_t;
7  typedef void* (*thread_run_func_t)(thread_run_data_t run_data);
8
9  class Thread{
10     pthread_t thread;
11     thread_run_func_t run_func;
12     thread_run_data_t run_data;
13 public:
14     Thread(thread_run_func_t run_func, thread_run_data_t run_data);
15     void destroy();
16     void start();
17     void join(void** result);
18     virtual ~Thread();
19     static void* starter(void* args);
20 };
21
22 #endif /*THREAD_H_*/

```

sep 20, 16 17:36

Thread.cpp

Page 1/1

```

1  #include "Thread.h"
2  #include <iostream> //cout
3
4  void* Thread::starter(void* args){
5     Thread* thread = (Thread*)args;
6     thread->run_func(thread->run_data);
7     return NULL;
8  }
9
10 Thread::Thread(thread_run_func_t run_func, thread_run_data_t run_data) {
11     this->run_func = run_func;
12     this->run_data = run_data;
13 }
14
15 void Thread::destroy(){
16 }
17
18 void Thread::start() {
19     //this es el dato
20     //starter es la funcion
21     pthread_create(&this->thread, NULL, starter, this);
22 }
23
24 void Thread::join(void** result){
25     pthread_join(this->thread,result);
26 }
27
28 Thread::~Thread(){}

```

sep 20, 16 17:36

Position.h

Page 1/1

```

1  #ifndef POSITION_H_
2  #define POSITION_H_
3
4  #include <iostream> //cout
5
6
7  class Position{
8      int column,row;
9  public:
10     Position(int rowNew,int columNew);
11     Position relativityPosition(Position& position);
12     Position sum(Position& position);
13     int getColumn();
14     int getRow();
15     void setColumn(int columnNew);
16     void setRow(int rowNew);
17     void print();
18     ~Position();
19 };
20 #endif /* POSITION_H_ */

```

sep 20, 16 17:36

Position.cpp

Page 1/1

```

1  #include "Position.h"
2
3
4
5  Position::Position(int rowNew,int columNew){
6      this->row = rowNew;
7      this->column = columNew;
8  }
9
10
11     //MÃtodo que te devuelve posicion relativa.
12     Position Position::relativityPosition(Position& otherPosition){
13         int columnNew = otherPosition.getColumn() - this->column;
14         int rowNew = otherPosition.getRow() - this->row;
15         Position newPosition(rowNew,columnNew);
16         return newPosition;
17     }
18
19     Position Position::sum(Position& position){
20         int newColumn = this->column + position.getColumn();
21         int newRow = this->row + position.getRow();
22         Position newPosition(newRow,newColumn);
23         return newPosition;
24     }
25
26     void Position::setColumn(int columnNew){
27         this->column = columnNew;
28     }
29
30     void Position::print(){
31         std::cout << "("<< this->row << "," << this->column << ")" << std::endl;
32     }
33
34     void Position::setRow(int rowNew){
35         this->row = rowNew;
36     }
37
38     int Position::getColumn(){
39         return column;
40     }
41
42     int Position::getRow(){
43         return row;
44     }
45
46     Position::~Position(){}

```

sep 20, 16 17:36

Matrix.h

Page 1/1

```

1  #ifndef MATRIX_H_
2  #define MATRIX_H_
3
4  #include "Position.h"
5  #include <vector>
6  #include <stdlib.h> // malloc, free
7  #include <string>
8
9  class Matrix{
10 private:
11     bool columnPositionValid(int column);
12     bool rowPositionValid(int row);
13     int cantRows, cantColumns;
14     std::string** matrix; /*puntero a la matriz*/
15 public:
16     Matrix(int rows, int column);
17     Matrix(const Matrix& matrix);
18     int getCantColumns() const;
19     int getCantRows() const;
20     std::string getElementPos(int posColumn, int posRows) const;
21     std::string getElementPos(Position position) const;
22     void setElementPos(int posRows, int posColumn, std::string element);
23     bool positionIsValid(Position& position);
24     void set(const Matrix& otherMatrix);
25     void dimesions();
26     void print();
27     ~Matrix();
28 };
29
30 #endif /* MATRIX_H_ */

```

sep 20, 16 17:36

Matrix.cpp

Page 1/2

```

1  #include "Matrix.h"
2  #include <string>
3
4  using std::string;
5  using std::cout;
6  using std::endl;
7
8  Matrix::Matrix(int rows, int column):cantRows(rows),cantColumns(column){
9      this->matrix = new string*[cantRows];
10     for (int i = 0; i < rows ; i++){
11         this->matrix[i] = new string[cantColumns];
12     }
13 }
14
15 Matrix::Matrix(const Matrix& otherMatrix):cantRows(otherMatrix.getCantRows()),
16 cantColumns(otherMatrix.getCantColumns()){
17     //cout << "asignation per copy" << endl;
18     this->matrix = new string*[cantRows];
19     for (int i = 0; i < cantRows ; i++){
20         this->matrix[i] = new string[cantColumns];
21     }
22     int i, j;
23     for (i = 1; i ≤ cantRows; i++) {
24         for (j = 1; j ≤ cantColumns; j++) {
25             this->setElementPos(i, j, otherMatrix.getElementPos(i, j));
26         }
27     }
28 }
29
30 void Matrix::set(const Matrix& otherMatrix){
31     if (otherMatrix.getCantColumns() == this->cantColumns &
32         otherMatrix.getCantRows() == this->cantRows){
33         for (int i = 1; i ≤ cantRows; i++){
34             for (int j = 1; j ≤ cantColumns; j++) {
35                 this->setElementPos(i, j, otherMatrix.getElementPos(i, j));
36             }
37         }
38     }
39     else{
40         cout << "no se puede copiar los valores" << endl;
41     }
42 }
43
44 //verifica si el num de columna es valido
45 bool Matrix::columnPositionValid(int column){
46     return 0 < column & column ≤ this->cantColumns;
47 }
48
49 //verifica si el num de fila es valido
50 bool Matrix::rowPositionValid(int row){
51     return 0 < row & row ≤ this->cantRows;
52 }
53
54 //muestra por stdout las dimensiones de la matrix
55 void Matrix::dimesions(){
56     cout << "Tengo col:" << this->cantColumns << "y fil:" << this->cantRows << endl;
57 }
58
59 void Matrix::print(){
60     for (int i = 0; i < this->cantRows; i++) {
61         for (int j = 0; j < this->cantColumns; j++) {
62             cout << matrix[i][j] << " ";
63         }
64         cout << " " << endl;
65     }
66 }

```

sep 20, 16 17:36

Matrix.cpp

Page 2/2

```

67 //cout << "+++++" << endl;
68 }
69
70 //verifica si la posicion fila,column es valida
71 bool Matrix::positionIsValid(Position& position){
72     return columnPositionValid(position.getColumn()) ^
73     rowPositionValid(position.getRow());
74 }
75 //-----getters-----
76 int Matrix::getCantColumns() const {
77     return cantColumns;
78 }
79
80 int Matrix::getCantRows() const{
81     return cantRows;
82 }
83
84 void Matrix::setElementPos(int posRows, int posColumn, string element){
85     matrix[posRows-1][posColumn-1] = element;
86 }
87
88 string Matrix::getElementPos(int posRows,int posColumn) const{
89     return matrix[posRows-1][posColumn-1];
90 }
91
92 string Matrix::getElementPos(Position position) const{
93     return matrix[position.getRow()-1][position.getColumn()-1];
94 }
95
96
97 Matrix::~Matrix(){
98     //cout << "destructor called" << endl;
99     for (int i = 0; i < cantRows; i++) {
100         delete[] matrix[i];
101     }
102     delete[] (matrix);
103 }

```

sep 20, 16 17:36

Interpreter.h

Page 1/1

```

1  #ifndef INTERPRETER_H_
2  #define INTERPRETER_H_
3
4  #include <iostream> //cout
5  #include <sstream>
6  #include <string>
7  #include <vector>
8
9  #include "Matrix.h"
10
11 class Interpreter{
12 private:
13     void split(const std::string &s, char delim, std::vector<std::string> &elems);
14 public:
15     Interpreter();
16     Matrix createMatrix(const std::string& matrix);
17     Matrix createMatrix(std::vector<std::string> elems);
18     ~Interpreter();
19 };
20 #endif /* INTERPRETER_H_ */

```

sep 20, 16 17:36

Interpreter.cpp

Page 1/2

```

1  #include "Interpreter.h"
2  #include <string>
3  #include <vector>
4  //Esta se encarga de interpretar una cadena y devolver la matriz
5  //saca el espacio redundante al principio pero no en el intermedio
6  using std::string;
7  using std::stoi;
8  using std::vector;
9  using std::stringstream;
10
11  Interpreter::Interpreter(){
12      //std::cout << "Soy una dilatacion" << std::endl;
13  }
14
15  Interpreter::~Interpreter(){}
16
17  Matrix Interpreter::createMatrix(const string& matrix){
18      //std::cout << "createMatrix" << std::endl;
19      vector<string> elems;
20      split(matrix, ' ', elems);
21      vector<string>::iterator it;
22      string::size_type sz;
23      it = elems.begin();
24      int row = stoi(*it,&sz);
25      ++it;
26      int colum = stoi(*it,&sz);
27      Matrix patron(row,colum);
28      //std::cout << row << " " << colum << std::endl;
29      int i = 0;
30      ++it;
31      for (; it != elems.end() ; ++it) {
32          string fila = *it;
33          for (int j = 0 ; j < colum ; j++) {
34              string elemento = fila.substr(j,1);
35              //std::cout << "Elemento:" << elemento << std::endl;
36              patron.setElementPos(i+1,j+1,elemento);
37          }
38          i++;
39      }
40      return patron;
41  }
42
43  Matrix Interpreter::createMatrix(std::vector<string> elems){
44      //std::cout << "createMatrix from vector" << std::endl;
45      string::size_type sz;
46      vector<string>::iterator it = elems.begin();
47      int row = std::stoi(*it,&sz);
48      ++it;
49      int colum = std::stoi(*it,&sz);
50      Matrix patron(row,colum);
51      //std::cout << row << " " << colum << std::endl;
52      int i = 0;
53      ++it;
54      for (; it != elems.end() ; ++it) {
55          string fila = *it;
56          for (int j = 0 ; j < colum ; j++) {
57              string elemento = fila.substr(j,1);
58              //std::cout << "Elemento:" << elemento << std::endl;
59              patron.setElementPos(i+1,j+1,elemento);
60          }
61          i++;
62      }
63      return patron;
64  }
65
66  void Interpreter::split(const string &s, char delim, vector<string> &elems) {

```

sep 20, 16 17:36

Interpreter.cpp

Page 2/2

```

67      stringstream ss;
68      ss.str(s);
69      string item;
70      while (getline(ss, item, delim)) {
71          elems.push_back(item);
72      }
73  }

```

sep 20, 16 17:36

FiltrosMorfologicos.cpp

Page 1/2

```

1  #include <iostream> //cout
2  #include <string> //compare
3  #include <string.h> //compare
4  #include <vector> //vector
5  #include "Position.h"
6  #include "Matrix.h"
7  #include "Interpreter.h"
8  #include "Dilatation.h"
9  #include "Erosion.h"
10
11 using std::string;
12 using std::cout;
13 using std::endl;
14 using std::cin;
15 using std::vector;
16
17 Filter* identifierFilter(string& filterString){
18     string dilatationString("d");
19     if (!filterString.compare(dilatationString)){
20         Dilatation* dilatation = new Dilatation();
21         return dilatation;
22     }
23     Erosion* erosion = new Erosion();
24     return erosion;
25 }
26
27 Matrix getMatrix(char* matrix){
28     Interpreter interpreter;
29     string matrixString(matrix);
30     Matrix oneMatrix = interpreter.createMatrix(matrixString);
31     return oneMatrix;
32 }
33 //      0      1      2      3      4      5
34 // ./tp <numero de hilos> <filtro 1> <patron 1> <filtro 2> <patron 2> ...
35 int main(int argc, char *argv[]) {
36     Interpreter interpreter;
37     //int cantThreads = atoi(argv[1]);
38     vector<string> vectorImagen;
39     //cout << "Cantidad de hilos:" << cantThreads << endl;
40     string input_line;
41     if (argc < 2){
42         cout << "Falta argumentos" << endl;
43         return 1;
44     }
45     if (cin) {
46         getline(cin, input_line);
47         size_t pos = input_line.find(" ");
48         //cout << pos << endl;
49         string col = input_line.substr(0,pos);
50         string row = input_line.substr(pos+1);
51         vectorImagen.push_back(row);
52         vectorImagen.push_back(col);
53     }
54     while (cin){
55         getline(cin, input_line);
56         if (input_line.compare("\n") == 1){
57             //std::cout << input_line << std::endl;
58             vectorImagen.push_back(input_line);
59         }
60     }
61     Matrix matrixOrigin = interpreter.createMatrix(vectorImagen);
62     Matrix& image(matrixOrigin);
63     //image.print();
64     for (int i = 2; i ≤ argc-2; i++) {
65         string filterString(argv[i]);
66         Filter* filter = identifierFilter(filterString);

```

sep 20, 16 17:36

FiltrosMorfologicos.cpp

Page 2/2

```

67     string matrixString(argv[i+1]);
68     Matrix patron = interpreter.createMatrix(matrixString);
69     //std::cout << "patron" << endl;
70     //patron.print();
71     Matrix resultado = filter->aplicateFilter(image,patron);
72     image.set(resultado);
73     //image.print();
74     i++;
75 }
76 std::cout <<image.getCantColumns()<< " "<<image.getCantRows()<< std::endl;
77 image.print();
78 return 0;
79 }

```

sep 20, 16 17:36

Filter.h

Page 1/1

```

1  #ifndef FILTER_H_
2  #define FILTER_H_
3
4  #include "Position.h"
5  #include "Matrix.h"
6  #include <list>
7
8  class Filter{
9  private:
10     std::list<bool> compareMatrices(Matrix& imagen,Matrix& patron,Position& pos);
11     virtual bool checkCoincidence(std::list<bool> lista) = 0;
12     Matrix createImageDestin(int row,int column);
13 public:
14     Filter();
15     virtual Matrix aplicateFilter(Matrix& image,Matrix& patron);
16     ~Filter();
17 };
18
19 #endif /* FILTER_H_ */

```

sep 20, 16 17:36

Filter.cpp

Page 1/2

```

1  #include "Filter.h"
2  #include <string>
3  #include <list>
4
5  using std::list;
6  using std::string;
7  //Constructor
8  Filter::Filter(){}
9
10 //Destructor
11 Filter::~Filter(){}
12
13 /*Pre:Recibe una matrix imagen y patron y una posicion en la cual se debe
14 comparar
15 Post:Devuelve una lista con todos los elementos que se compararon
16 */
17 list<bool> Filter::compareMatrices(Matrix& imagen,Matrix& patron,Position& pos){
18     list<bool> lista;
19     int row = patron.getCantRows();
20     int column = patron.getCantColumns();
21     Position posicionMedia(row/2 + 1, column/2 + 1);
22     Position posicionRelativa = posicionMedia.relativityPosition(pos);
23     Position otherPosition(0,0);
24     string asterisco("#");
25     bool valor;
26     int i,j;
27     for (i = 1; i ≤ row; i++) {
28         for (j = 1; j ≤ column; j++) {
29             otherPosition.setRow(i);
30             otherPosition.setColumn(j);
31             Position posImagen = posicionRelativa.sum(otherPosition);
32             //Posicion no valida
33             if (imagen.positionIsValid(posImagen) == 0){
34                 lista.push_back(false);
35             }else{
36                 //posicion valida
37                 if (asterisco.compare(patron.getElementPos(i,j)) == 0){
38                     string elemento = imagen.getElementPos(posImagen);
39                     valor = elemento.compare(asterisco);
40                     if (valor == 0){
41                         lista.push_back(true);
42                     }else{
43                         lista.push_back(false);
44                     }
45                 }
46             }
47         }
48     }
49     return lista;
50 }
51
52 Matrix Filter::createImageDestin(int row,int column){
53     Matrix destino(row,column);
54     for (int i = 1; i ≤ row; i++) {
55         for (int j = 1; j ≤ column; j++) {
56             destino.setElementPos(i,j,".");
57         }
58     }
59     return destino;
60 }
61
62 Matrix Filter::aplicateFilter(Matrix& image,Matrix& patron){
63     Position pivote(0,0);
64     int row = image.getCantRows();

```

sep 20, 16 17:36

Filter.cpp

Page 2/2

```

67  int column = image.getCantColumns();
68  Matrix pepe = createImageDestin(row,column);
69  for (int i = 1; i ≤ row; i++){
70      for (int j = 1; j ≤ column; j++){
71          pivote.setRow(i);
72          pivote.setColumn(j);
73          list<bool> lista = compareMatrices(image,patron,pivote);
74          bool valor = checkCoincidence(lista);
75          if (valor){
76              pepe.setElementPos(pivote.getRow(),pivote.getColumn(),"#");
77          }
78      }
79  }
80  //std::cout << "-----" << std::endl;
81  return pepe;
82  }

```

sep 20, 16 17:36

Erosion.h

Page 1/1

```

1  #ifndef EROSION_H_
2  #define EROSION_H_
3  #include "Filter.h"
4  #include <list>
5
6  class Erosion : public Filter{
7  private:
8      bool checkCoincidence(std::list<bool> lista);
9  public:
10     Erosion();
11     ~Erosion();
12 };
13
14
15 #endif /* EROSION_H_ */

```


sep 20, 16 17:36

Erosion.cpp

Page 1/1

```

1  #include "Erosion.h"
2  #include <list>
3  using std::list;
4
5  Erosion::Erosion(){
6      //std::cout << "Soy una erosion" << std::endl;
7  }
8
9  Erosion::~Erosion(){}
10
11
12 //Cuando se aplica erosion se necesita chequear si hay coincidencia total
13 //Pre: Recibe un list de valores booleanos
14 //Post: Chequea si existe coincidencia total
15 bool Erosion::checkCoincidence(std::list<bool> lista){
16     //std::cout << "checkCoincidence erosion" << std::endl;
17     list<bool>::iterator it;
18     for (it=lista.begin(); it != lista.end(); ++it){
19         bool valor = *it;
20         //std::cout << valor << " ";
21         if (!valor){
22             return false;
23         }
24     }
25     return true;
26 }

```

sep 20, 16 17:36

Dilatation.h

Page 1/1

```

1  #ifndef DILATATION_H_
2  #define DILATATION_H_
3  #include "Filter.h"
4  #include <list>
5
6  class Dilatation : public Filter{
7  private:
8      bool checkCoincidence(std::list<bool> lista);
9  public:
10     Dilatation();
11     ~Dilatation();
12 };
13
14
15 #endif /* DILATATION_H_ */

```

sep 20, 16 17:36	Dilatation.cpp	Page 1/1
1	<code>#include "Dilatation.h"</code>	
2		
3	<code>#include <list> //list</code>	
4	<code>using std::list;</code>	
5		
6	<code>Dilatation::Dilatation(){} </code>	
7		
8	<code>Dilatation::~Dilatation(){} </code>	
9		
10		
11	<code>//Cuando se aplica erosion se necesita chequear si hay coincidencia total</code>	
12	<code>//Pre:Recibe un list de valores booleanos</code>	
13	<code>//Post: Chequea si existe coincidencia parcial</code>	
14	<code>bool Dilatation::checkCoincidence(std::list<bool> lista){</code>	
15	<code> //std::cout << "Dilatation checkCoincidence" << std::endl;</code>	
16	<code> list<bool>::iterator it;</code>	
17	<code> for (it=lista.begin(); it != lista.end(); ++it){</code>	
18	<code> bool valor = *it;</code>	
19	<code> if (valor){</code>	
20	<code> return true;</code>	
21	<code> }</code>	
22	<code> }</code>	
23	<code> return false;</code>	
24	<code>}</code>	

sep 20, 16 17:36	Table of Content	Page 1/1
1	Table of Contents	
2	1 <i>Thread.h</i> sheets 1 to 1 (1) pages 1- 1 23 lines	
3	2 <i>Thread.cpp</i> sheets 1 to 1 (1) pages 2- 2 29 lines	
4	3 <i>Position.h</i> sheets 2 to 2 (1) pages 3- 3 21 lines	
5	4 <i>Position.cpp</i> sheets 2 to 2 (1) pages 4- 4 47 lines	
6	5 <i>Matrix.h</i> sheets 3 to 3 (1) pages 5- 5 31 lines	
7	6 <i>Matrix.cpp</i> sheets 3 to 4 (2) pages 6- 7 104 lines	
8	7 <i>Interpreter.h</i> sheets 4 to 4 (1) pages 8- 8 21 lines	
9	8 <i>Interpreter.cpp</i> sheets 5 to 5 (1) pages 9- 10 74 lines	
10	9 <i>FiltrosMorfologicos.cpp</i> sheets 6 to 6 (1) pages 11- 12 80 lines	
11	10 <i>Filter.h</i> sheets 7 to 7 (1) pages 13- 13 21 lines	
12	11 <i>Filter.cpp</i> sheets 7 to 8 (2) pages 14- 15 83 lines	
13	12 <i>Erosion.h</i> sheets 8 to 8 (1) pages 16- 16 16 lines	
14	13 <i>Erosion.cpp</i> sheets 9 to 9 (1) pages 17- 17 27 lines	
15	14 <i>Dilatation.h</i> sheets 9 to 9 (1) pages 18- 18 16 lines	
16	15 <i>Dilatation.cpp</i> sheets 10 to 10 (1) pages 19- 19 25 lines	