75:42 - Taller de Programación I

Ejercicio Nº 2 Padrón 94719

Alumno GONZÁLEZ, CRISTIAN Firma

Nota:	3 (tercera)	Corrige:	Di Paolo	Entrega #1
- no pasa ningún caso de prueba - Ver correcciones en el código				Fecha de entrega Fecha de devolución

Nota:		Corrige:		Entrega #2
				Fecha de entrega Fecha de devolución

El presente trabajo, así como la entrega electrónica correspondiente al mismo, constituyen una obra de creación completamente personal, no habiendo sido inspirada ni siendo copia completa o parcial de ninguna fuente pública, privada, de otra persona o naturaleza.

```h
#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_*/
```

Ya no se usa pthread en la cátedra sino que se usa la implementación de threads de C++11


```cpp
#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(){}
```

```h
#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_ */
```



```cpp
#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(){}
```

```h
#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_ */
```

un array de punteros a un array de std::string donde cada string contendrá un solo caracter !! muy complejo

en general
- los objetos pueden no deben ser copiables. Si llegase necesario, implementar un método copy explícito
- deben ser movible

así, en el que estoy trabajando


```cpp
#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;
  }
```

```
37    //cout << "++++++++++++++" << endl;
38  }
39
70  //verifica si la posicion fila,colum 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++) {
00      delete[] matrix[i];
01    }
02    delete[](matrix);
03  }
```

por referencia constante


```
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_ */
```

```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) {
```

muy complejo, C++ puede parsearlo por vos con sstream o sstringstream

código duplicado


```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);
```

Código de debug


```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;
}
```

No usé hilos

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

debe se virtual


```
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
53 Matrix Filter::createImageDestin(int row,int column){
54   Matrix destino(row,column);
55   for (int i = 1; i ≤ row; i++) {
56     for (int j = 1; j ≤ column; j++) {
57       destino.setElementPos(i,j,".");
58     }
59   }
60   return destino;
61 }
62
63
64 Matrix Filter::aplicateFilter(Matrix& image,Matrix& patron){
65   Position pivote(0,0);
66   int row = image.getCantRows();
```

```
57    int column = image.getCantColumns();
58    Matrix pepe = createImageDestin(row,column);
59    for (int i = 1; i ≤ row; i++){
60      for (int j = 1; j ≤ column; j++){
61        pivote.setRow(i);
62        pivote.setColumn(j);
63        list<bool> lista = compareMatrices(image,patron,pivote);
64        bool valor = checkCoincidence(lista);
65        if (valor){
66          pepe.setElementPos(pivote.getRow(),pivote.getColumn(),"#");
67        }
68      }
69    }
70    //std::cout << "-------------------------" << std::endl;
71    return pepe;
72  }
```



```
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_ */
```

```cpp
#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;
}
```



```cpp
#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_ */
```