Escuela de computación
Cartago
Arquitectura de computadores
IC-3101
Esteban Arias Méndez
II semestre 2017
19/06/17
Proyecto #3
Simulador CPU
Carlos Gomes Osa, 2016121385
Jeremy Live González, 2015068507

✓ Abstract:

On this project we introduce you on the process of operation with matrix. We applicant the distributed processing, parallelism, time-sharing and concurrency and we implement correctly apply high-level programming language instructions that make use of tools to complement concurrency and parallelism such as the use of threads (threads) among others to provide an adequate response to the problems highlighted.

Pasos a seguir, para poder ejecutar el código :

1- Se debe de introducir las matrices con las que se quieres trabajar, se pueden hasta n. La aplicación es introducirlas en una pila.
2- Se introduce la operación matricial que se quiere efectuar y lo siguiente que hace el programa es hacer 2 pop de la PILA(FIFO) y operar dichas matrices en base a los hilos(Threads) que el usuario digita previamente.
3- En Base a los hilos(Threads) se divide el problema que se efectuara.
4- Se podrá navegar en la matriz resultante con 4 botones integrados.
5- El usuario puede ingresar y/o modificar los parámetros del programa en cada iteración.
6- Interfaz muy interactiva para el usuario ☺

✓ Introducción :

En este proyecto programado de Matrices se trabajó en el lenguaje de alto nivel C++ y en lenguaje de bajo nivel NASM.

Se desarrolló un programa que permite al usuario aplicar operaciones entre matrices. Estas operaciones a implementar son: Suma de matrices, Producto de un escalar por una matriz, Producto de matrices, Matriz Inversa, Calculo de la transpuesta de una matriz.

Para cada matriz resultante se indica si el tipo de la misma es: matriz fila, matriz columna, matriz rectangular, matriz cuadrada, matriz nula, Triangular superior, Triangular inferior, matriz diagonal, matriz escalar, matriz identidad o unidad.

Se utilizó una LIGA para poder ejecutar en él .cpp archivos .asm, esto para lograr que el programa en alto nivel solamente colecte los datos, los transmite al programa NASM y con el resultado desde NASM desplegarlos al usuario. Se escogió la operación ESCALAR POR MATRIZ para ser implementada completamente en NASM.

✓ **Desarrollo :**

Lógica para efectuar las operaciones entre matrices en base a los hilos que digito el usuario. Se decidió hacerse del siguiente modo:

1 – Para el caso de multiplicación de matrices se hace lo siguiente:

1: Se divide el hilo equitativamente. En cada hilo se agarra una fila(matriz1) y una columna(matriz2) se hace las operaciones y se escribe en un archivo con el hilo para escribir.

2: Esto disminuye el tiempo en la creación del archivo y la aritmética de operaciones

2 – Para el caso de suma de matrices se hace lo siguiente:

1: Se divide el hilo equitativamente. En cada hilo se agarra una fila(matriz1) y una fila(matriz2) se hace la suma con respecto a su índice.

2: Se escribe en un archivo con el hilo para escribir.

3 – Para el caso de escalar se efectúa en un hilo el escalar y en el otro se va cambiando las fila de la matriz.

4 – En el caso de verificar que matriz es, se hace en alto nivel.

✓ **Análisis :**

Los hilos funcionan de la siguiente forma:

1 – Existe un hilo lector el cual reparte el trabajo. Por ejemplo si el usuario digita 4 hilos el lector le reparte a los cuatro tan equitativamente como puede.

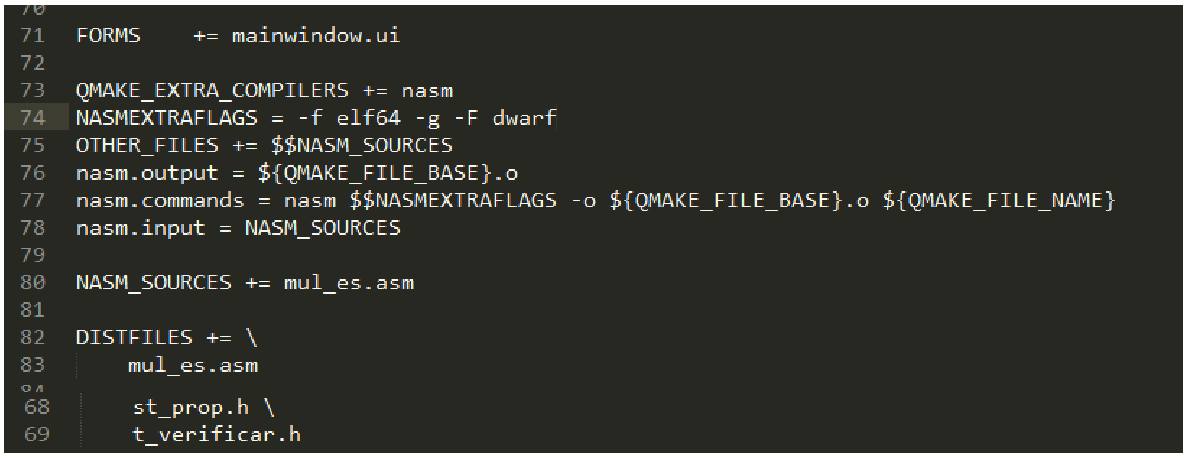
2 – Existe también un hilo escritor, que recibe los resultados de los hilos que hacen operaciones, y lo escribe en el documento resultante.

La LIGA entre C++ y NASM se efectuó de la siguiente forma:

1 – En el archivo "Matriz.pro" se debe de poner la línea

mul_es.asm

Ósea esto para que pueda compilar el IDE QT con el .asm



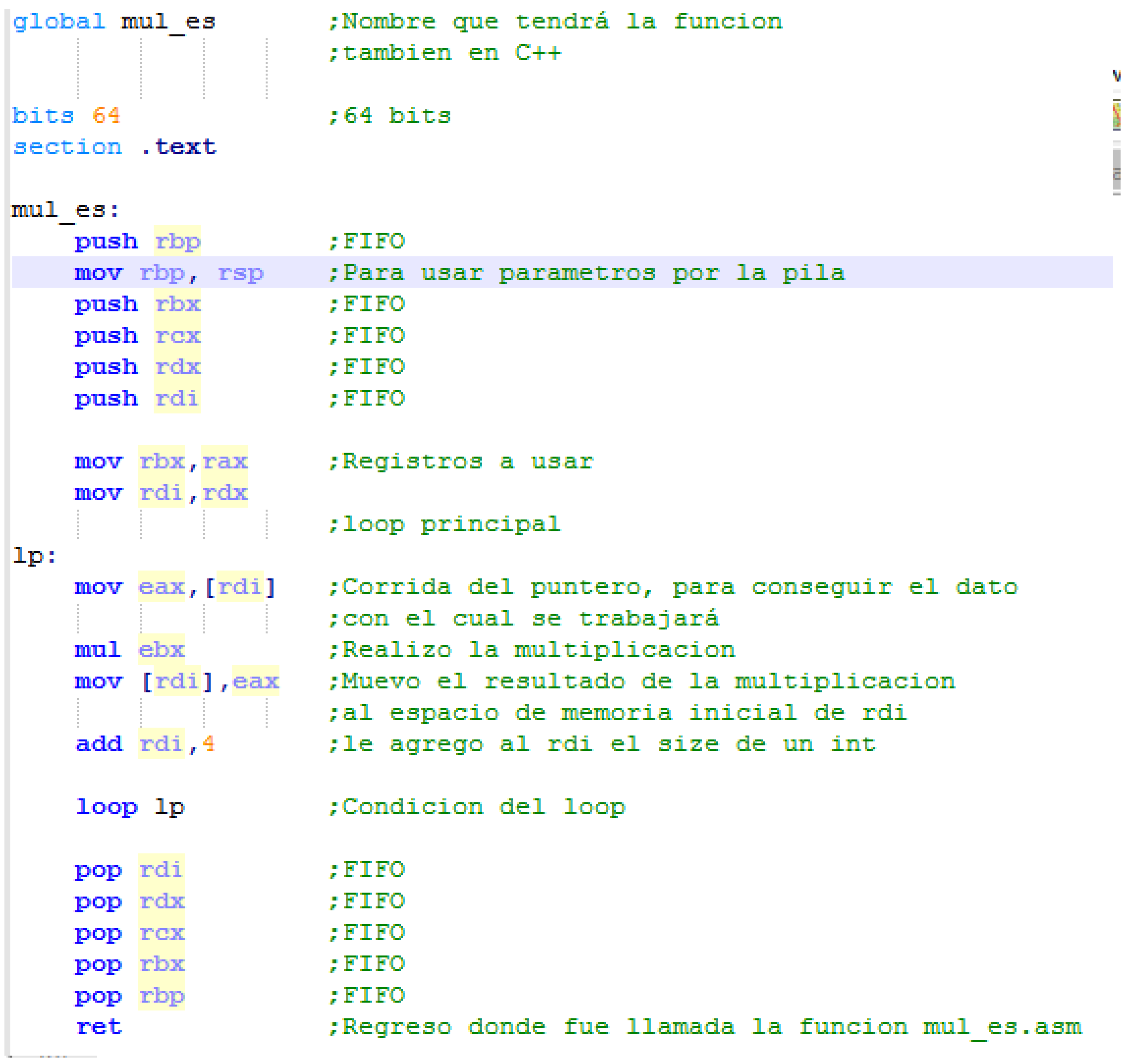
```
FORMS    += mainwindow.ui

QMAKE_EXTRA_COMPILERS += nasm
NASMEXTRAFLAGS = -f elf64 -g -F dwarf
OTHER_FILES += $$NASM_SOURCES
nasm.output = ${QMAKE_FILE_BASE}.o
nasm.commands = nasm $$NASMEXTRAFLAGS -o ${QMAKE_FILE_BASE}.o ${QMAKE_FILE_NAME}
nasm.input = NASM_SOURCES

NASM_SOURCES += mul_es.asm

DISTFILES += \
    mul_es.asm
    st_prop.h \
    t_verificar.h
```

2 – En el archivo "Mul_es.asm" se realizó la operación de producto escalar por matriz:



```
global mul_es           ;Nombre que tendrá la funcion
                        ;tambien en C++

bits 64                 ;64 bits
section .text

mul_es:
    push rbp            ;FIFO
    mov rbp, rsp        ;Para usar parametros por la pila
    push rbx            ;FIFO
    push rcx            ;FIFO
    push rdx            ;FIFO
    push rdi            ;FIFO

    mov rbx,rax         ;Registros a usar
    mov rdi,rdx
                        ;loop principal
lp:
    mov eax,[rdi]       ;Corrida del puntero, para conseguir el dato
                        ;con el cual se trabajará
    mul ebx             ;Realizo la multiplicacion
    mov [rdi],eax       ;Muevo el resultado de la multiplicacion
                        ;al espacio de memoria inicial de rdi
    add rdi,4           ;le agrego al rdi el size de un int

    loop lp             ;Condicion del loop

    pop rdi             ;FIFO
    pop rdx             ;FIFO
    pop rcx             ;FIFO
    pop rbx             ;FIFO
    pop rbp             ;FIFO
    ret                 ;Regreso donde fue llamada la funcion mul_es.asm
```

{

✓ Conclusión :

En este tercer proyecto programa se aprendió a trabajar con hilos en alto nivel y a conectar C++ con NASM por medio de una liga.

Se aprendió a trabajar en equipo.

Fue complementario programar este proyecto.

Muy didáctico.

✓ Apéndices :

```
----------------File_manager.cpp
#include "file_manager.h"
#include <QIODevice>
#include <QDebug>
file_manager::file_manager()
{
}
QFile* file_manager::createFile(QString s)
{
    QFile* file = new QFile(s);
    if (!file->open(QIODevice::ReadOnly | QIODevice::Text | QIODevice::ReadWrite))
    {
        qDebug() << "FAIL TO CREATE FILE / FILE NOT EXIT***";
    }
    return file;
}
----------file_manager.h
#ifndef FILE_MANAGER_H
#define FILE_MANAGER_H
#include "QString"
#include "QFile"
class file_manager
{
public:
    file_manager();
    QFile* createFile(QString s);
private:
};
#endif // FILE_MANAGER_H
-----------file_thread.cpp
#include "file_thread.h"
#include <QDebug>
#include <QtGlobal>
#include <QTime>
#include <QByteArray>
file_thread::file_thread(QObject *parent) : QThread(parent)
{
     abort = false;
     restart = false;
}
file_thread::~file_thread()
{
}
void file_thread::w_document(QFile *file, int x, int y, int max, int min)
{
        x_s = x;
        y_s = y;
        ma = max;
        mi = min;
        working = file;
        start();
}
void file_thread::run()
{
    qsrand(QTime::currentTime().msec());
    long int total = x_s * y_s;
    QString r;
    QByteArray s_s;
    char cell_size = 1;
    s_s.push_back(char(x_s/16777216));
    s_s.push_back(char(x_s/65536));
    s_s.push_back(char(x_s/256));
    s_s.push_back(char(x_s%256));
    s_s.push_back(char(y_s/16777216));
    s_s.push_back(char(y_s/65536));
    s_s.push_back(char(y_s/256));
    s_s.push_back(char(y_s%256));
    s_s.push_back(cell_size);
    working->write(s_s);
    for(long int i = 0; i < total; i++){
        if(i%x_s == 0){
            emit progress((100*i)/total);
        }
        r = QString((unsigned char)(qrand()%(ma+1-mi) + mi));
        /*while(r.size() < max_n_size){
            r = QString(" ").append(r);
        }*/
        working->write(r.toUtf8());
    }
    working->resize(working->pos());
    emit n_m();
    return;
}
--------------------file_thread.h
#ifndef FILE_THREAD_H
#define FILE_THREAD_H
#include "QThread"
#include "QFile"
#include "QObject"
class file_thread : public QThread
{
    Q_OBJECT
public:
    file_thread(QObject *parent = 0);
    ~file_thread();
    void w_document(QFile *file, int x, int y, int max, int min);
private:
    int x_s, y_s, ma, mi;
    bool restart;
    bool abort;
    QFile *working;
protected:
    void run();
signals:
    void n_m();
    void progress(int);
};

#endif // FILE_THREAD_H
-------------------file_thread_constructor.cpp
#include "file_thread_constructor.h"
#include "math.h"
#include "QDebug"
file_thread_constructor::file_thread_constructor()
{
    remaining = new QVector<line*>();
    remaining_bytes = new QVector<st_byte*>();
    mg = new file_manager;
    paused = false;
}
void file_thread_constructor::w_set_up(QPoint s, int c_s, int thr, QFile *r)
{
    size = QPoint(s.x(),s.y());
    cell_size = c_s;
    running = true;
    res = r;
    r_t = thr;
}
QByteArray file_thread_constructor::get_arr(int n,int cs)
{
    QByteArray r;
    int tn = n;
    char * c = new char[cs];
    for(int i  =0; i < cs; i++){
        c[i] =(char)(tn%256);
        tn/=256;
    }
    r = QByteArray(c,cs);
    return r;
}

void file_thread_constructor::on_line(int *l, int line_number)
{
    //qDebug()<<"got line";
    QMutex lock;
    lock.lock();
    remaining->push_back(new line(l,line_number));
    lock.unlock();
}
void file_thread_constructor::on_byte(int byte, QPoint place)
{
    QMutex lock;
    lock.lock();
    remaining_bytes->push_back(new st_byte(byte,place.x(),place.y()));
    lock.unlock();
}
//codigo de trabajo en el thread
void file_thread_constructor::run()
{
    qDebug()<<"running,  file size: "<<size.x()*size.y()*cell_size+9;
    //se reserve el espacio necesario para guardar la matriz
    res->resize(size.x()*size.y()*cell_size+9);
    line *l;
    st_byte* b;
    //se colocan los datos del archivo en el header
    QByteArray s_s;
    s_s.push_back(char(size.x()/16777216));
    s_s.push_back(char(size.x()/65536));
    s_s.push_back(char(size.x()/256));
    s_s.push_back(char(size.x()%256));
    s_s.push_back(char(size.y()/16777216));
    s_s.push_back(char(size.y()/65536));
    s_s.push_back(char(size.y()/256));
    s_s.push_back(char(size.y()%256));
    s_s.push_back((char)cell_size);
    res->seek(0);
    res->write(s_s);
    QByteArray to_write;
    to_write.resize(size.x()*cell_size);
    qDebug()<<"Writter waiting";
    while(running){
        //ser procesa una linea a escribir
        if(remaining->size() > 0){
            l = remaining->at(remaining->size()-1);
            remaining->pop_back();
            for(int i = 0; i < size.x();i++){
                for(int j = 0; j < cell_size; j++){
                    if(j)
                    to_write[i*cell_size+j] = (l->content[i]/256*(cell_size-j));
                    else
                    to_write[i*cell_size+j] = (l->content[i]%256);
                }
            }
            res->seek(9 + l->line_number*size.x()*cell_size);
            res->write(to_write);
        }else if(remaining_bytes->size() > 0){
            b = remaining_bytes->at(remaining_bytes->size()-1);
            remaining_bytes->pop_back();
            QByteArray ba = get_arr(b->data,cell_size);
            res->seek(9+b->y*cell_size*size.x() + b->x);
            res->write(ba);//escribe un byte
            /*delete b;*/
            //qDebug()<<"writing byte: "<<r_t;
        }else if(r_t <= 0){
            qDebug()<<"Respuesta Guardada";
            emit finished();
            break;
        }
    }
```

```
}
void file_thread_constructor::stop()
{
    r_t--;
}
---------------------file_thread_constructor
#ifndef FILE_THREAD_CONSTRUCTOR_H
#define FILE_THREAD_CONSTRUCTOR_H

#include <QObject>
#include <QThread>
#include <QString>
#include <QFile>
#include <QVector>
#include <QPoint>
#include <st_line.h>
#include <st_byte.h>
#include <file_manager.h>
#include <QByteArray>
#include <QMutex>

class file_thread_constructor : public QThread
{
    Q_OBJECT
public:
    file_thread_constructor();
    void w_set_up(QPoint s, int c_s, int thr, QFile* r);
    QByteArray get_arr(int n, int cs);
public slots:
    void on_line(int *l, int line_number);
    void on_byte(int byte, QPoint place);
    void stop();
protected:
    void run();
signals:
    void finished();
    void r_pause();
    void r_cont();
private:
    bool paused;
    QPoint size;
    int cell_size;
    QVector<line*>* remaining;
    QVector<st_byte*>* remaining_bytes;
    bool running;
    QFile* res;
    file_manager* mg;
    int r_t;
};
#endif // FILE_THREAD_CONSTRUCTOR_H
-------------------------list_display.cpp
#include "list_display.h"
#include "QDebug"
#include "QPoint"
#include "QApplication"
#include <QSignalMapper>
#include <QGraphicsProxyWidget>


List_Display::List_Display(QGraphicsScene *sc)
{
    //guardo mi referencia a la clase contenedora
    scene = sc;
    content = new QVector<QString>();
    titulos = new QVector<QPushButton*>();
    //mapper, encargado de las señales de los botones

    x_space = 100;
    y_space = 20;

    //label vacia para que se centre el objeto
    QLabel *lb = new QLabel();
    QGraphicsProxyWidget * proxy = scene->addWidget(lb);
    proxy->setPos(0,0);
    lb->move(600,650);
    lb->setStyleSheet("QLabel{background-color: rgba(0,0,0,0)}");

    //segundo label vacio
    QLabel *lb2 = new QLabel();
    QGraphicsProxyWidget * proxy2 = scene->addWidget(lb);
    proxy2->setPos(0,0);
    lb2->move(0,0);
    lb2->setStyleSheet("QLabel{background-color: rgba(0,0,0,0)}");




}

//agrega un elemento al contenido del objeto
void List_Display::add_element(QString string)
{
      content->push_back(string);
      actualizar();

}

//actualiza la interfaz
void List_Display::actualizar()
{
    int total;
    for(total = 0; total < content->size(); total++){
        //si no existe el label lo creo

        if(titulos->size() < total + 1){
            QPushButton *tmp = new QPushButton();

            //tmp->setParent(vista);
            //agrego el boton a la scena
            QGraphicsProxyWidget * proxy = scene->addWidget(tmp);
            proxy->setPos(0,0);
        }

            tmp->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color: 
rgba(0,0,53,190);font-size: 20px}");



            //tamaño fijo del boton
            tmp->setFixedWidth(80);
            tmp->setFixedHeight(80);

            //asigna posicion del boton
            tmp->move(x_space*total + 30, y_space);

            titulos->push_back(tmp);

            //mapper excesivo, mapea las señales a una id
            QSignalMapper *mapper = new QSignalMapper(this);
            connect(mapper,SIGNAL(mapped(int)),this,SLOT(onButton(int)));


            //conecto la señal del boton al ser liberado
            connect(tmp,SIGNAL(released()),mapper,SLOT(map()));
            mapper->setMapping(tmp,total);




        }

        if(titulos->at(total)){
        titulos->at(total)->setText(content->at(total));
        titulos->at(total)->setVisible(true);
        }

    }

    while(total < titulos->size()){
        titulos->at(total)->setVisible(false);
        total++;
    }
```

```
}

void List_Display::onButton(int id)
{

    //qDebug()<<"clicked: "<<id;
    emit(on_button(id));
}
--------------------list_display.cpp
#include "list_display.h"
#include "QDebug"
#include "QPoint"
#include "QApplication"
#include <QSignalMapper>
#include <QGraphicsProxyWidget>
List_Display::List_Display(QGraphicsScene *sc)
{
    //guardo mi referencia a la clase contenedora
    scene = sc;
    content = new QVector<QString>();
    titulos = new QVector<QPushButton*>();
    //mapper, encargado de las señales de los botones

    x_space = 100;
    y_space = 20;

    //label vacia para que se centre el objeto
    QLabel *lb = new QLabel();
    QGraphicsProxyWidget * proxy = scene->addWidget(lb);
    proxy->setPos(0,0);
    lb->move(600,650);
    lb->setStyleSheet("QLabel{background-color: rgba(0,0,0,0)}");

    //segundo label vacio
    QLabel *lb2 = new QLabel();
    QGraphicsProxyWidget * proxy2 = scene->addWidget(lb);
    proxy2->setPos(0,0);
    lb2->move(0,0);

    lb2->setStyleSheet("QLabel{background-color: rgba(0,0,0,0)}");





}


//agrega un elemento al contenido del objeto

void List_Display::add_element(QString string)

{

      content->push_back(string);

      actualizar();


}


//actualiza la interfaz

void List_Display::actualizar()

{

    int total;

    for(total = 0; total < content->size(); total++){

        //si no existe el label lo creo


        if(titulos->size() < total + 1){

            QPushButton *tmp = new QPushButton();


            //tmp->setParent(vista);

            //agrego el boton a la scena

            QGraphicsProxyWidget * proxy = scene->addWidget(tmp);

            proxy->setPos(0,0);

            tmp->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color:
rgba(0,0,53,190);font-size: 20px}");




            //tamaño fijo del boton

            tmp->setFixedWidth(80);

            tmp->setFixedHeight(80);
}

            //asigna posicion del boton
            tmp->move(x_space*total + 30, y_space);

            titulos->push_back(tmp);

            //mapper excesivo, mapea las señales a una id
            QSignalMapper *mapper = new QSignalMapper(this);
            connect(mapper,SIGNAL(mapped(int)),this,SLOT(onButton(int)));



            //conecto la señal del boton al ser liberado
            connect(tmp,SIGNAL(released()),mapper,SLOT(map()));
            mapper->setMapping(tmp,total);




        }

        if(titulos->at(total)){
        titulos->at(total)->setText(content->at(total));
        titulos->at(total)->setVisible(true);
        }

    }

    while(total < titulos->size()){
        titulos->at(total)->setVisible(false);
        total++;
    }
}

void List_Display::onButton(int id)
{

    //qDebug()<<"clicked: "<<id;
    emit(on_button(id));
}
```

```
------------------------list_display.h
#ifndef LIST_DISPLAY_H
#define LIST_DISPLAY_H

#include <QObject>
#include <QString>
#include <QGraphicsView>
#include <QVector>
#include <QLabel>
#include <QPushButton>
#include <QGraphicsScene>




class List_Display : public QObject{
    Q_OBJECT
public:
    List_Display(QGraphicsScene * sc);
    QGraphicsScene * scene;
    void add_element(QString string);
    void actualizar();

private:
    QVector<QString> * content;
    QVector<QPushButton*> * titulos;
    int x_space;
    int y_space;

signals:
    void on_button(int);
private slots:
    void onButton(int id);
};

#endif // LIST_DISPLAY_H
-----------------------main.cpp
```

```
#include <QApplication>
#include "screen.h"
#include <QObject>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    screen *ventana = new screen();
    ventana->mostrar();
    QObject::connect(&a,SIGNAL(aboutToQuit()),ventana,SLOT(closeEvent()));

    return a.exec();
}
---------------mul_es.asm
global mul_es                   ;Nombre que tendrá la funcion
                                                    ;tambien en C++

bits 64                                   ;64 bits
section .text

mul_es:
    push rbp                    ;FIFO
    mov rbp, rsp   ;Actualizo el dato de rbp
    push rbx                    ;FIFO
    push rcx                    ;FIFO
    push rdx                    ;FIFO
    push rdi                    ;FIFO

    mov rbx,rax                 ;Registros a usar
    mov rdi,rdx
                                                    ;loop principal
lp:
    mov eax,[rdi]  ;Corrida del puntero, para conseguir el dato
                                                    ;con el cual se trabajará
    mul ebx                               ;Realizo la multiplicacion
    mov [rdi],eax  ;Muevo el resultado de la multiplicacion
                                                    ;al espacio de memoria inicial de rdi
    add rdi,4                    ;le agrego al rdi el size de un int

    loop lp                             ;Condicion del loop

    pop rdi                             ;FIFO
    pop rdx                             ;FIFO
    pop rcx                             ;FIFO
    pop rbx                             ;FIFO
    pop rbp                             ;FIFO
    ret                                 ;Regreso donde fue llamada la funcion mul_es.asm
---------------------operator.cpp
#include "operator.h"
#include <QString>
#include <QStringList>
#include <QDebug>
#include <cmath>
#include <QByteArray>
#include <QPoint>

Operator::Operator()
{
    corriendo = 0;
    r = mg->createFile("answer.m");
    a_c_s = 1;
}

QFile *Operator::sumar(QFile *f1, QFile *f2, int thr)
{

    //se cargan los datos de los archivos
    f1->seek(0);
    f2->seek(0);
    QByteArray h1 = f1->read(9);
    QByteArray h2 = f2->read(9);
    m1 = sToInt(h1);
    m2 = sToInt(h2);
    //revisa si ambas matrices miden lo mismo
    if(m1 != m2)
    {   emit finished_op();
        return 0;


    }
    int size_m1 = h1.at(8);
    int size_m2 = h2.at(8);


    //revisa si las matrices son del mismo tamaño
    //if(size_m1 != size_m2)return 0;


    qDebug()<<"Sumando matrices\nTamaño:  "<<m1.x()<<", "<<m1.y()<<"  matriz2:"<<m2.x()<<",
"<<m2.y();




    t_r_suma* repartidor = new t_r_suma();
    repartidor->set_up(f1,f2,m1,QPoint(size_m1,size_m2),thr);


    //thread escritor
    file_thread_constructor* w = new file_thread_constructor();


    if(size_m1 > size_m2){
        w->w_set_up(m1, size_m1,thr,r);
    }else{
        w->w_set_up(m1, size_m2,thr,r);
    }
    connect(w,SIGNAL(finished()),this, SLOT(onF_op()));
    w->start();


    for(int i = 0; i < thr;i++){
        t_sumador* sumador = new t_sumador();
        sumador->set_up(i,m1,QPoint(size_m1,size_m2));
        connect(sumador,SIGNAL(n_w(int)),repartidor,SLOT(ask_work(int)));

connect(repartidor,SIGNAL(work(QByteArray,QByteArray,int,int)),sumador,SLOT(add_work(QByteArray,QBy
teArray,int,int)));
        connect(sumador,SIGNAL(done(int*,int)),w,SLOT(on_line(int*,int)));
        connect(repartidor,SIGNAL(end()),sumador,SLOT(onEnd()));
        connect(sumador,SIGNAL(finished()),sumador,SLOT(deleteLater()));
        connect(sumador,SIGNAL(finished()),w,SLOT(stop()));
        connect(repartidor,SIGNAL(progres(double)),this,SLOT(onProgres(double)));
        sumador->start();

    }
    repartidor->start();
    /*
    //se genera el thread que guarda archivos
    file_thread_constructor* w = new file_thread_constructor();

    if(size_m1 > size_m2){
        w->w_set_up(m1, size_m1,thr,r);
    }else{
        w->w_set_up(m1, size_m2,thr,r);
    }
    w->start();

    qDebug()<<"archivo creado";

    //se generan los threads que operan
    for(int i = 0; i < thr;i++){
        QThread *thread = new QThread;
        w_sumador *s = new w_sumador();
        s->addWork(f1,f2,m1,i,thr,size_m1,size_m2);

        s->moveToThread(thread);
        //connect(s, SIGNAL(error(QString)), this, SLOT(errorString(QString)));
        connect(thread, SIGNAL(started()), s, SLOT(process()));
        connect(s, SIGNAL(finished()), thread, SLOT(quit()));
        connect(s, SIGNAL(finished()), s, SLOT(deleteLater()));
        connect(thread, SIGNAL(finished()), thread, SLOT(deleteLater()));
        connect(s,SIGNAL(done(int*,int)),w,SLOT(on_line(int*,int)));
        connect(s,SIGNAL(w_done()),w,SLOT(stop()));
        connect(w,SIGNAL(r_pause()),s,SLOT(pause()));
        connect(w,SIGNAL(r_cont()),s,SLOT(cont()));
        connect(w,SIGNAL(finished()),s,SLOT(destroy()));
```

```
        qDebug()<<"corriendo suma  "<<i;

        thread->start();

    }

    qDebug()<<"terminado";

    */

    return r;



}


QFile *Operator::sumar_escalar(QFile *f1, int n,int c_s, int thr)

{   //se cargan los datos de los archivos

    f1->seek(0);

    QByteArray h1 = f1->read(9);

    m1 = sToInt(h1);

    //revisa si ambas matrices miden lo mismo

    //alrevez ya que estan en la pila

    int size_m1 = h1.at(8);


    //revisa si las matrices son del mismo tamaño

    //if(size_m1 != size_m2)return 0;


    qDebug()<<"Multiplicando matrices\nTamaño:  "<<m1.x()<<", "<<m1.y()<<"  matriz2:"<<m2.x()<<",
"<<m2.y();


    //thread escritor

    file_thread_constructor* w = new file_thread_constructor();



    w->w_set_up(m1, size_m1,thr,r);



    connect(w,SIGNAL(finished()),this, SLOT(onF_op()));

    w->start();


    t_r_mul_es* repartidor = new t_r_mul_es();

    repartidor->set_up(f1,c_s,size_m1,m1.x(),thr);
    for(int i  = 0; i < thr; i ++){
        t_mult_escalar* multiplicador =  new t_mult_escalar();
        multiplicador->set_up(n,size_m1,c_s,m1.x(),i);
connect(repartidor,SIGNAL(work(QByteArray,int,int)),multiplicador,SLOT(onWork(QByteArray,int,int)));
        connect(multiplicador,SIGNAL(finished()),w,SLOT(stop()));
        connect(multiplicador,SIGNAL(finished()),multiplicador,SLOT(deleteLater()));
        connect(multiplicador,SIGNAL(done(int*,int)),w,SLOT(on_line(int*,int)));
        connect(repartidor,SIGNAL(finished()),multiplicador,SLOT(onEnd()));
        connect(repartidor,SIGNAL(progres(double)),this,SLOT(onProgres(double)));
        multiplicador->start();
    }



    repartidor->start();

}

QFile *Operator::multiplicar(QFile *f1, QFile *f2, int thr)
{
    //se cargan los datos de los archivos
    f1->seek(0);
    f2->seek(0);
    QByteArray h1 = f1->read(9);
    QByteArray h2 = f2->read(9);
    m1 = sToInt(h1);
    m2 = sToInt(h2);
    //revisa si ambas matrices miden lo mismo
    //alrevez ya que estan en la pila
    if(m1.y() != m2.x()){
        emit finished_op();
    return 0;}
    int size_m1 = h1.at(8);
    int size_m2 = h2.at(8);

    //revisa si las matrices son del mismo tamaño
    //if(size_m1 != size_m2)return 0;


    qDebug()<<"Multiplicando matrices\nTamaño:  "<<m1.x()<<", "<<m1.y()<<"  matriz2:"<<m2.x()<<",
"<<m2.y();


    t_r_multiplicador* repartidor = new t_r_multiplicador();

    repartidor->set_up(f2,f1,m2,m1,QPoint(size_m2,size_m1),thr);


    //thread escritor

    file_thread_constructor* w = new file_thread_constructor();


    if(size_m1 > size_m2){

        w->w_set_up(m1, size_m1,thr,r);

    }else{

        w->w_set_up(m1, size_m2,thr,r);

    }

    connect(w,SIGNAL(finished()),this, SLOT(onF_op()));

    w->start();


    //codigo de los threads multiplicadores


    for(int i  = 0; i < thr; i++){

        t_multiplicador* multiplicador = new t_multiplicador();

        multiplicador->set_up(i,m1.x(),QPoint(size_m1,size_m2));

connect(repartidor,SIGNAL(work(QByteArray,QByteArray,int,int,int)),multiplicador,SLOT(work(QByteArray,Q
ByteArray,int,int,int)));

        connect(multiplicador,SIGNAL(byte(int,QPoint)),w,SLOT(on_byte(int,QPoint)));

        connect(multiplicador,SIGNAL(finished()),multiplicador,SLOT(deleteLater()));

        connect(multiplicador,SIGNAL(finished()),w,SLOT(stop()));

        connect(repartidor,SIGNAL(end()),multiplicador,SLOT(n_m_w()));

        connect(repartidor,SIGNAL(progres(double)),this,SLOT(onProgres(double)));

        multiplicador->start();

    }


    repartidor->start();


    return r;
```

```
}

QFile *Operator::transpuesta(QFile *f1,int c_s)
{
    //se cargan los datos de los archivos
    f1->seek(0);
    QByteArray h1 = f1->read(9);
    m1 = sToInt(h1);
    //revisa si ambas matrices miden lo mismo
    //alrevez ya que estan en la pila
    int size_m1 = h1.at(8);
    //revisa si las matrices son del mismo tamaño
    //if(size_m1 != size_m2)return 0;

    qDebug()<<"inviertiendo matriz\nTamaño:  "<<m1.x()<<", "<<m1.y();

    t_transpuesta* tran = new t_transpuesta();
    tran->set_up(f1,r,m1,c_s,size_m1);
    connect(tran,SIGNAL(finished()),this,SLOT(onF_op()));
    connect(tran,SIGNAL(finished()),tran,SLOT(deleteLater()));
    int x =m1.x();
    int y = m1.y();
    m1.setX(y);
    m1.setY(x);
    tran->start();



    return r;
}

QFile *Operator::rango(QFile *f1)
{

}

void Operator::verficar(QFile *f1)
{
}

//retorna las caracteristicas de una matriz leyendo los primeros 8 bytes
QPoint Operator::sToInt(QByteArray s)
{
    QPoint re;
    //qDebug()<<"convirtiendo  "<<s.size();
    int r = 0;
    for(int i  = 0; i < 4; i++){
        r += ((unsigned char)s.at(i))*(pow(256,(3-i)));
        //qDebug()<<((unsigned char)s.at(i))*(pow(256,(3-i)));
    }
    re.setX(r);
    r = 0;
    for(int i  = 4; i < 8; i++){
        r += ((unsigned char)s.at(i))*(pow(256,(7-i)));
    }
    re.setY(r);

    return re;

}

void Operator::n_f()
{
    r = mg->createFile("answer.m");
}

QPoint *Operator::getSize()
{
    QPoint* n = new QPoint(m1.x(),m1.y());
    return n;
}

QFile *Operator::get_res()
{
    return r;
}

void Operator::onProgres(double n)
{
    emit progres(n);
}

void Operator::onF_op()
{
    emit finished_op();
}
--------------operator.h
#ifndef OPERATOR_H
#define OPERATOR_H

#include <QFile>
#include <QThread>
#include <w_sumador.h>
#include <w_multiplicador.h>
#include <QObject>
#include <w_writer.h>
#include <file_thread_constructor.h>
#include <file_manager.h>
#include <t_r_suma.h>
#include <t_sumador.h>
#include <t_r_multiplicador.h>
#include <t_multiplicador.h>
#include <t_transpuesta.h>
#include "t_r_mul_es.h"
#include "t_mult_escalar.h"

class Operator:public QObject
{
    Q_OBJECT
public:
    Operator();

    QFile *sumar(QFile* f1,QFile* f2, int thr);
    QFile* sumar_escalar(QFile* f1, int n, int c_s, int thr);
    QFile* multiplicar(QFile *f1, QFile *f2, int thr);
    QFile* transpuesta(QFile* f1, int c_s);
    QFile* rango(QFile* f1);
    void verficar(QFile* f1);//verifica el tipo de matriz
    QPoint sToInt(QByteArray s);
    void n_f();
    QPoint* getSize();
    QFile* get_res();
private:
    int corriendo;
    QFile *r;
    file_manager* mg;
    QPoint m1;
    QPoint m2;
    int a_c_s;
private slots:
    void onProgres(double n);
    void onF_op();
signals:
    void progres(double);
    void finished_op();
};

#endif // OPERATOR_H
------------------screen.cpp
#include "screen.h"
#include <QBrush>
#include <QColor>
#include <QDebug>
#include <QtGlobal>
#include <QTime>
#include <QString>
#include <QFont>
```

```
#include <QGraphicsProxyWidget>

#include <QScrollBar>




screen::screen()

{

    //variables---------------------------------------

    contador = 0;

    selected = -1;

    all_matrix = new QVector<QFile*>();

    all_matrix_size = new QVector<QPoint*>();

    qsrand(QTime::currentTime().msec());

    files = new file_manager();

    f_thread = new file_thread();

    f_x = f_y = 0;

    observando = false;


    //escena -----------------------------------------

    scene = new QGraphicsScene(this);

    QBrush fondo(QColor(00,0,00,255));

    scene->setBackgroundBrush(fondo);



    //view -------------------------------------------

    container = new QGraphicsView();



    //cambiar titulo de la ventana -------------------

    this->setWindowTitle("Matrix operator");








    int y_disp = -50;
    int x_disp = 20;

    unsigned int maxmimo = 2147483640;

    //Elementos para crear una matriz----------------------------------------

    l_fondo_crear = new QLabel();

    l_fondo_crear->setParent(container);

    l_fondo_crear->setStyleSheet("QLabel{background-color: rgba(0,0,53,190); border: 2px solid}");

    l_fondo_crear->setFixedSize(480,350);

    l_fondo_crear->move(10 + x_disp,300 + y_disp);

    /*l_fondo_n = new QLabel();

    l_fondo_n->setParent(container);

    l_fondo_n->setStyleSheet("QLabel{background-color: rgba(120,120,200,80); border: 1px solid}");

    l_fondo_n->setFixedSize(1600,180);

    l_fondo_n->move(0,20);*/


    b_crear = new QPushButton(container);

    b_crear->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color:
rgba(0,0,53,190);font-size: 20px}");

    b_crear->move(180 + x_disp,560 + y_disp);

    b_crear->setText("Crear matriz");


    //labels

    l_titulo_crear = new QLabel("Crear una nueva matriz");

    l_titulo_crear->setStyleSheet("QLabel{color: rgb(60,180,60);font-size: 30px}");

    l_titulo_crear->setParent(container);

    l_titulo_crear->move(60 + x_disp,330 + y_disp);



    int line1 = 460 + y_disp;

    int line2 = 500 + y_disp;

    l_min = new QLabel("Minimo :");

    l_max = new QLabel("Maximo:");

    l_min->setParent(container);

    l_max->setParent(container);

    l_min->setStyleSheet("QLabel{color: rgb(70,150,70);font-size: 25px}");

    l_max->setStyleSheet("QLabel{color: rgb(70,150,70);font-size: 25px}");

    l_min->move(50 + x_disp, line1);

    l_max->move(50 + x_disp, line2);
    l_x_size = new QLabel("Ancho:");

    l_y_size = new QLabel("Alto   :");

    l_x_size->setParent(container);

    l_y_size->setParent(container);

    l_x_size->setStyleSheet("QLabel{color: rgb(70,150,70);font-size: 25px}");

    l_y_size->setStyleSheet("QLabel{color: rgb(70,150,70);font-size: 25px}");

    l_x_size->move(260 + x_disp, line1);

    l_y_size->move(260 + x_disp, line2);

    //spin boxes

    sb_min = new QSpinBox(container);

    sb_min->setMinimum(-100);

    sb_min->setMaximum(0);

    sb_min->setValue(0);

    sb_min->move(160 + x_disp, line1);

    sb_min->setFixedHeight(30);

    sb_min->setFixedWidth(80);

    sb_min->setStyleSheet("QSpinBox{color: rgb(255,255,255);background-color: rgba(0,120,0,60);font-
size: 15px}");


    sb_max= new QSpinBox(container);

    sb_max->setMinimum(1);

    sb_max->setMaximum(100);

    sb_max->setValue(1);

    sb_max->move(160 + x_disp, line2);

    sb_max->setFixedHeight(30);

    sb_max->setFixedWidth(80);

    sb_max->setStyleSheet("QSpinBox{color: rgb(255,255,255);background-color: rgba(0,120,0,60);font-
size: 15px}");


    sb_x_size = new QSpinBox(container);

    sb_x_size->setMinimum(1);

    sb_x_size->setMaximum(maxmimo);

    sb_x_size->setValue(10);

    sb_x_size->move(370+ x_disp,line1);

    sb_x_size->setFixedHeight(30);

    sb_x_size->setFixedWidth(80);
    sb_x_size->setStyleSheet("QSpinBox{color: rgb(255,255,255);background-color: rgba(0,120,0,60);font-
size: 15px}");


    sb_y_size= new QSpinBox(container);

    sb_y_size->setMinimum(1);

    sb_y_size->setMaximum(maxmimo);

    sb_y_size->setValue(10);

    sb_y_size->move(370 + x_disp, line2);

    sb_y_size->setFixedHeight(30);

    sb_y_size->setFixedWidth(80);

    sb_y_size->setStyleSheet("QSpinBox{color: rgb(255,255,255);background-color: rgba(0,120,0,60);font-
size: 15px}");


    //name

    l_name = new QLabel("Nombre:");

    l_name->setParent(container);

    l_name->move(x_disp+50,y_disp+400);

    l_name->setStyleSheet("QLabel{color: rgb(70,150,70);font-size: 25px}");


    te_name = new QLineEdit();

    te_name->setText("0");

    te_name->setParent(container);

    te_name->move(x_disp+160,y_disp+400);

    te_name->setStyleSheet("QLineEdit{color: rgb(255,255,255);background-color: rgba(0,120,0,60);font-
size: 18px}");


    pb_crear = new QProgressBar (container);

    pb_crear->move(50 + x_disp, 570 + y_disp);

    pb_crear->setFixedSize(400,40);

    pb_crear->setVisible(false);





    //botones de opciones

    b_editar = new QPushButton("editar",container);

    b_observar = new QPushButton("Observar",container);

    b_operar = new QPushButton("Operar",container);
    b_editar->move(30, 140);

    b_observar->move(150, 140);

    b_operar->move(280, 140);

    b_editar->setStyleSheet("QPushButton{background-color: rgba(0,80,10,120);font-size: 18px}");

    b_observar->setStyleSheet("QPushButton{background-color: rgba(0,120,25,120);font-size: 18px}");

    b_operar->setStyleSheet("QPushButton{background-color: rgba(0,140,60,120);font-size: 18px}");


    QFont font;

    font.setFamily("Courier");

    font.setStyleHint(QFont::Monospace);

    font.setFixedPitch(true);

    font.setPointSize(10);




    //elementos de operar--------------------------------------------------

    x_disp = 0;

    y_disp = 0;


    b_sumar = new QPushButton("Sumar",container);

    b_multiplicar = new QPushButton("Multiplicar",container);

    b_transpuesta = new QPushButton("Transpuesta",container);

    b_sumar_escalar = new QPushButton("Mult escalar",container);

    b_rango = new QPushButton("Rango",container);

    b_salvar = new QPushButton("Salvar Ultima respuesta",container);

    b_remove = new QPushButton("Remover",container);

    b_sumar->move(60+x_disp,400+y_disp);

    b_multiplicar->move(120+x_disp,480+y_disp);

    b_transpuesta->move(190+x_disp,400+y_disp);

    b_sumar_escalar->move(250+x_disp,480+y_disp);

    b_rango->move(320+x_disp,400+y_disp);

    b_salvar->move(750+x_disp,450+y_disp);

    b_remove->move(680+x_disp,600+y_disp);

    b_sumar->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color: 
rgba(10,53,0,60);font-size: 18px}");

    b_multiplicar->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color: 
rgba(10,53,0,60);font-size: 18px}");

    b_transpuesta->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color: 
rgba(10,53,0,60);font-size: 20px}");
```

```
    b_sumar_escalar->setStyleSheet("QPushButton{color: rgb(255,255,255);background-color:
rgba(10,53,0,60);font-size: 18px}");

    b_rango->setStyleSheet("QPushButton{color: rgb(150,150,150);background-color:
rgba(10,10,10,60);font-size: 18px}");

    b_sumar->setFixedSize(130,60);

    b_multiplicar->setFixedSize(130,60);

    b_transpuesta->setFixedSize(130,60);

    b_sumar_escalar->setFixedSize(130,60);

    b_rango->setFixedSize(130,60);


    pb_operar = new QProgressBar(container);

    pb_operar->move(x_disp+105,y_disp+300);

    pb_operar->setFixedSize(300,50);

    pb_operar->setVisible(false);


    sb_threads = new QSpinBox(container);

    sb_cell_size = new QSpinBox(container);

    sb_threads->setMinimum(1);

    sb_threads->setMaximum(1000);

    sb_cell_size->setMinimum(1);

    sb_cell_size->setMaximum(4);

    sb_cell_size->move(780+x_disp,200+y_disp);

    sb_threads->move(780+x_disp,250+y_disp);

    sb_cell_size->setFixedWidth(80);

    sb_threads->setFixedWidth(80);

    l_n_threads = new QLabel("Threads: ",container);

    l_cell_Size = new QLabel("Tamaño(bytes):",container);

    l_n_threads->move(x_disp+680,y_disp+250);

    l_cell_Size->move(x_disp+680,y_disp+200);

    sb_threads->setVisible(false);

    sb_cell_size->setVisible(false);

    l_n_threads->setVisible(false);

    l_cell_Size->setVisible(false);


    sb_escal = new QSpinBox(container);

    sb_escal->setMinimum(-100);

    sb_escal->setMaximum(100);

    sb_escal->move(x_disp+380,y_disp+500);
    sb_escal->setStyleSheet("QSpinBox{background-color: rgba(40,80,40,30);font-size: 20;color:
rgba(130,130,130,150)}");

    sb_escal->setValue(5);



    le_name = new QLineEdit(container);

    le_name->move(765+x_disp,480+y_disp);


    l_elementos = new QLabel(container);





    operandos = new v_display(container,600+x_disp,630+y_disp);


    operar_visibilidad(false);

    operando  =false;


    names = new QVector<QString>();

    pending_names = new QVector<QString>();

    pendientes = new QVector<int>();



    //elementos de observar matriz------------------------------------------

    m_disp = new v_matix_Display(container);

    m_disp->set_visibility(false);

    l_prop = new QLabel(container);

    l_prop->setStyleSheet("QLabel{font-size: 15px}");

    l_prop->move(450,120);

    l_prop->setFixedSize(700,50);


    //operador---------------------------------------------------------------

    op = new Operator();


    //objeto que contiene las matrices---------------------------------------

    matrices = new List_Display(scene);

    /*matrices->add_element(QString("Gato"));

    matrices->add_element(QString("Perro"));
    matrices->add_element(QString("Avion"));*/


    //se asigna la scena ----------------------------------------------------
    container->setScene(scene);
    container->setMinimumSize(640,480);



    //signals y slots---------------------------------------------------------
    connect(matrices,SIGNAL(on_button(int)),this,SLOT(on_l_button(int)));
    connect(b_crear,SIGNAL(released()),this,SLOT(onCrear()));
    connect(b_observar,SIGNAL(released()),this,SLOT(onObservar()));
    connect(b_editar,SIGNAL(released()),this,SLOT(onEditar()));
    connect(f_thread,SIGNAL(progress(int)),this,SLOT(onProgressMatrix(int)));
    connect(f_thread,SIGNAL(n_m()),this,SLOT(onNewMatrix()));
    connect(b_operar,SIGNAL(released()),this,SLOT(onOperar()));
    connect(b_sumar,SIGNAL(released()),this,SLOT(onSumar()));
    connect(b_multiplicar,SIGNAL(released()),this,SLOT(onMultiplicar()));
    connect(b_sumar_escalar,SIGNAL(released()),this,SLOT(onMultiplicar_e()));
    connect(b_transpuesta,SIGNAL(released()),this,SLOT(onTrans()));
    connect(b_salvar,SIGNAL(released()),this,SLOT(onSalvar()));
    connect(op,SIGNAL(progres(double)),this,SLOT(onProgresOp(double)));
    connect(op,SIGNAL(finished_op()),this,SLOT(onFinishedOp()));

}

void screen::mostrar()
{

    container->show();



}

void screen::on_l_button(int id)
{
    //qDebug()<<"matriz: "<<id<<"   x: "<<all_matrix_size->at(id)->x()<<"   y: "<<all_matrix_size-
>at(id)->y();
```

```
    if(operando){
        pendientes->push_back(id);
        pending_names->push_back(names->at(id));
        qDebug()<<"agregado: "<<names->at(id);
        operandos->actualizar(pending_names);
    }else if (observando){
        m_disp->actualize(all_matrix->at(id),all_matrix_size->at(id));
        emit clear();
        t_verificar * t = new t_verificar();
        t->set_up(all_matrix->at(id));

        connect(t,SIGNAL(res(QString)),this,SLOT(onProps(QString)));
        connect(this,SIGNAL(clear()),t,SLOT(on_clear()));
        connect(t,SIGNAL(finished()),t,SLOT(deleteLater()));
        t->start();}
}

void screen::onCrear()
{
    b_editar->setVisible(false);
    b_observar->setVisible(false);
    b_crear->setVisible(false);
    pb_crear->setVisible(true);
    b_operar->setVisible(false);

    container->horizontalScrollBar()->setSliderPosition(0);
    container->verticalScrollBar()->setSliderPosition(0);

    QFile* arr = files->createFile(te_name->text().append(".m"));
    all_matrix->push_back(arr);
    all_matrix_size->push_back(new QPoint(sb_x_size->value(),sb_y_size->value()));
    random_matrix(arr,sb_x_size->value(),sb_y_size->value(),sb_max->value(),sb_min->value());

    contador++;
    matrices->add_element(te_name->text());
    names->push_back(te_name->text());
    te_name->setText(QString::number(contador));
    le_name->setText(QString::number(contador));




}

void screen::onObservar()
{   m_disp->set_visibility(true);
    crear_visibilidad(false);
    operar_visibilidad(false);
    l_prop->setVisible(true);
    container->horizontalScrollBar()->setSliderPosition(0);
    container->verticalScrollBar()->setSliderPosition(0);
    operando = false;
    observando = true;
}

void screen::onEditar()
{
    l_prop->setVisible(false);
    m_disp->set_visibility(false);
    crear_visibilidad(true);
    operar_visibilidad(false);
    container->horizontalScrollBar()->setSliderPosition(0);
    container->verticalScrollBar()->setSliderPosition(0);
    operando = false;
    observando = false;
}

void screen::closeEvent()
{
    QFile *del;
    for(int i = 0; i < all_matrix->size(); i++){
        del = all_matrix->at(i);
        all_matrix->removeAt(i);
        delete del;
    }
}

void screen::onNewMatrix()
{
    b_crear->setVisible(true);
    pb_crear->setVisible(false);
    b_editar->setVisible(true);
    b_observar->setVisible(true);
    b_operar->setVisible(true);
}

void screen::onProgressMatrix(int v)
{
    pb_crear->setValue(v);
}

void screen::onOperar()
{
    emit clear();
    m_disp->set_visibility(false);
    crear_visibilidad(false);
    operar_visibilidad(true);
    operando = true;
    observando = false;
    l_prop->setVisible(false);
}

void screen::onSumar()
{
    if(pendientes->size()>1){
        pb_operar->setVisible(true);
        op_vis(false);
        op->sumar(all_matrix->at(pendientes->at(pendientes->size()-1)),all_matrix->at(pendientes-
>at(pendientes->size()-2)),sb_threads->value());
        pendientes->pop_back();
        pendientes->pop_back();
        pending_names->pop_back();

        pending_names->pop_back();

        operandos->actualizar(pending_names);


    }

}


void screen::onMultiplicar()

{

    if(pendientes->size()>1){

        pb_operar->setVisible(true);

        op_vis(false);

        op->multiplicar(all_matrix->at(pendientes->at(pendientes->size()-1)),all_matrix->at(pendientes-
>at(pendientes->size()-2)),sb_threads->value());

        pendientes->pop_back();

        pendientes->pop_back();

        pending_names->pop_back();

        pending_names->pop_back();

        operandos->actualizar(pending_names);

    }

}


void screen::onMultiplicar_e()

{

    if(pendientes->size()>0){

        pb_operar->setVisible(true);

        op_vis(false);

        op->sumar_escalar(all_matrix->at(pendientes->at(pendientes->size()-1)),sb_escal-
>value(),sb_cell_size->value(),sb_threads->value());

        pendientes->pop_back();

        pending_names->pop_back();

        operandos->actualizar(pending_names);


    }

}


void screen::onTrans()

{
    if(pendientes->size()>0){
        pb_operar->setVisible(true);
        op_vis(false);
        op->transpuesta(all_matrix->at(pendientes->at(pendientes->size()-1)),sb_cell_size->value());
        pendientes->pop_back();
        pending_names->pop_back();
        operandos->actualizar(pending_names);

    }
}

void screen::onSalvar()
{

    container->horizontalScrollBar()->setSliderPosition(0);
    container->verticalScrollBar()->setSliderPosition(0);

    QString name =le_name->text().append("r.m");
    //elimino el archivo si ya existe
    QFile::remove(name);

    QFile* arr = op->get_res();
    arr->rename(name);
    qDebug()<<"guardado como"<<arr->fileName();

    arr->close();

    arr = new QFile(name);
    qDebug()<<arr<<name;
    arr->open(QIODevice::ReadWrite| QIODevice::ReadOnly );
    all_matrix->push_back(arr);
    all_matrix_size->push_back(op->getSize());
    op->n_f();

    contador++;
    matrices->add_element(le_name->text().append("r"));
    names->push_back(le_name->text().append("r"));
    te_name->setText(QString::number(contador));
    le_name->setText(QString::number(contador));
}

void screen::onProgresOp(double p)
{
    pb_operar->setValue((int)p);

}

void screen::onFinishedOp()
{
    pb_operar->setVisible(false);
    op_vis(true);
}

void screen::onProps(QString s)
{
    //qDebug()<<s;
    l_prop->setText(s);
}



void screen::crear_visibilidad(bool v)
{
    b_crear->setVisible(v);
    sb_min->setVisible(v);
    sb_max->setVisible(v);
    sb_x_size->setVisible(v);
    sb_y_size->setVisible(v);
    l_min->setVisible(v);
    l_max->setVisible(v);
    l_x_size->setVisible(v);
    l_y_size->setVisible(v);
    l_titulo_crear->setVisible(v);
    l_name->setVisible(v);
    te_name->setVisible(v);
    l_fondo_crear->setVisible(v);
}

void screen::operar_visibilidad(bool v)
{
    b_sumar->setVisible(v);
    b_multiplicar->setVisible(v);
    b_transpuesta->setVisible(v);
    b_sumar_escalar->setVisible(v);
    b_rango->setVisible(v);

    b_salvar->setVisible(v);
    b_remove->setVisible(v);
    le_name->setVisible(v);

    l_elementos->setVisible(v);
    operandos->set_visible(v);

    sb_threads->setVisible(v);
    sb_cell_size->setVisible(v);
    l_n_threads->setVisible(v);
    l_cell_Size->setVisible(v);
    sb_escal->setVisible(v);

}

void screen::random_matrix(QFile *m, int x, int y, int max, int min)
{

    f_thread->w_document(m,x,y,max,min);



}

void screen::op_vis(bool v)
{
    b_sumar->setVisible(v);
    b_sumar_escalar->setVisible(v);
    b_transpuesta->setVisible(v);
    b_multiplicar->setVisible(v);
    b_rango->setVisible(v);
    b_editar->setVisible(v);
    b_observar->setVisible(v);
    b_operar->setVisible(v);
    b_salvar->setVisible(v);
    le_name->setVisible(v);
}
-------------------screen.h
#ifndef SCREEN_H
#define SCREEN_H

#include <QMainWindow>
#include <QObject>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <list_display.h>
#include <QPushButton>
#include <QLabel>
#include <QSpinBox>
#include <QLineEdit>
#include <QVector>
#include <QPoint>
#include <QTextEdit>
#include <file_manager.h>
#include <QFile>
#include <file_thread.h>
#include <QProgressBar>
#include "v_display.h"
#include "operator.h"
#include <v_matix_display.h>
#include <QTime>
#include <t_verificar.h>

class screen : public QMainWindow{
    Q_OBJECT
public:
    screen();
    void mostrar();
private slots:

public slots:
    void on_l_button(int id);
    void onCrear();
    void onObservar();
    void onEditar();
    void closeEvent();
    void onNewMatrix();
    void onProgressMatrix(int v);
    void onOperar();
    void onSumar();
    void onMultiplicar();
    void onMultiplicar_e();
    void onTrans();
    void onSalvar();
    void onProgresOp(double p);
    void onFinishedOp();
    void onProps(QString s);
signals:
    void clear();
private:
    void crear_visibilidad(bool v);
    void operar_visibilidad(bool v);
    void random_matrix(QFile *m, int x, int y, int max, int min);
    void op_vis(bool v);

    QGraphicsView *container;
    QGraphicsScene *scene;
    List_Display * matrices;

    //variables
    int contador;//cantidad de matrices existentes;
```

```
QVector<QFile*> *all_matrix;//vector que contiene todas las matrices
QVector<QPoint*> *all_matrix_size;//contiene los tamaños de todas las matrices
int selected;//indice de la matriz seleccionada actualmente
int f_x, f_y;

//elementos de crear matriz
QPushButton *b_crear;
QSpinBox *sb_min;
QSpinBox *sb_max;
QSpinBox *sb_x_size;
QSpinBox *sb_y_size;
QLabel *l_min;
QLabel *l_max;
QLabel *l_x_size;
QLabel *l_y_size;
QLabel *l_titulo_crear;
QLabel *l_name;
QLineEdit *te_name;
QLabel *l_fondo_crear;
QLabel *l_fondo_n;
file_manager *files;
file_thread *f_thread;
QProgressBar *pb_crear;


//opciones
QPushButton *b_observar;
QPushButton *b_editar;
QPushButton *b_operar;



//elemntos de operar
QPushButton *b_sumar;
QPushButton *b_multiplicar;
QPushButton *b_transpuesta;
QPushButton *b_sumar_escalar;
```

```
    QPushButton *b_rango;

    QPushButton *b_salvar;
    QPushButton *b_remove;
    QLineEdit *le_name;

    QProgressBar* pb_operar;
    v_display *operandos;
    QSpinBox* sb_threads;
    QSpinBox* sb_cell_size;
    QSpinBox* sb_escal;
    QLabel* l_n_threads;
    QLabel* l_cell_Size;
    QLabel* l_prop;

    QLabel *l_elementos;
    QVector<int> *pendientes;
    QVector<QString> *names;
    QVector<QString> *pending_names;
    bool operando;
    bool observando;

    Operator *op;

    //elementos de observar
    v_matix_Display* m_disp;

};

#endif // SCREEN_H
-------------------st_byte.h
#ifndef ST_BYTE_H
#define ST_BYTE_H

struct st_byte{
    int data;
    int x, y;
    st_byte(int c, int X, int Y){
        data = c;
        x = X;
        y = Y;
    }
};



#endif // ST_BYTE_H
--------------st_line.h
#ifndef ST_LINE_H
#define ST_LINE_H
struct line{
    int *content;
    int line_number;
    line(int* arr, int l){
        content = arr;
        line_number = l;
    }
};
#endif // ST_LINE_H
----------------st_prop.h
#ifndef ST_PROP_H
#define ST_PROP_H

struct s_prop{
    bool fila;
    bool col;
    bool cuadrada;
    bool rect;

    bool diag;
    bool nula;
    bool t_sup;
    bool t_inf;
    bool id;
    bool m_nul;
    s_prop(int x,int y){
        fila = col = cuadrada = rect = false;
        nula = t_sup = t_inf = id = diag = true;
        if(x == y){
            cuadrada = true;
        }else{
            rect = true;
            id = false;
            diag = false;
        }
        if(x == 1){
            col = true;
        }
        if(y == 1){
            fila = true;
        }

    }




};

#endif // ST_PROP_H
-------------t_mult_escalar.cpp
#include "t_mult_escalar.h"
#include "QDebug"
#include "math.h"

t_mult_escalar::t_mult_escalar()
{
    ending = false;
    pending = new QVector<node*>();
}

void t_mult_escalar::set_up(int n,int o_cs ,int cs, int x_s,int Id)
```

```
{

    escalar = n;
    n_cs = cs;
    c_s = o_cs;
    x = x_s;
    id = Id;
}

void t_mult_escalar::run()
{
    int*l1;
    l1 = new int[x];

    while(true){
        if(pending->size() >0){
            int *r = new int[x];

            //acomodo las lineas a procesar
            line1 = pending->at(pending->size()-1);





            get_array(r,line1->arr,c_s);









            //operacion en ensamblador
            mul_es(escalar,x,r);



            /*QString f;
            for(int i  = 0;  i< x; i++){
                f.append(QString::number(r[i])).append(" ");
            }
            f.append("  n: ").append(QString::number(escalar));
            qDebug()<<f;
            //qDebug()<<"line sent "<<lines->at(lines->size()-1);*/
            emit done(r,line1->l);
            pending->pop_back();

        }else{
            if(ending){

            break;}
        }



        }
        emit finished();
    }







void t_mult_escalar::get_array(int *arr, QByteArray* original, int c_s)
{
    for(int i = 0; i < original->size(); i+=c_s){
        arr[i] = 0;
        for(short int j = 0; j < c_s; j++){
            arr[i] += (int)(((unsigned char)original->at(i*c_s))*(pow(256,c_s-j-1)));
        }
    }
}

void t_mult_escalar::onWork(QByteArray arr, int n, int Id)
{
    if(Id == id){
    QByteArray* a = new QByteArray(arr,arr.size());
    node *t = new node(a, n);
    pending->push_back(t);}
}

void t_mult_escalar::onEnd()
{
    ending = true;
}
------------------t_mult_escalar.h
#ifndef T_MULT_ESCALAR_H
#define T_MULT_ESCALAR_H
#include "QThread"
#include "QObject"
#include "QFile"
#include "QPoint"
#include "QVector"
#include "QByteArray"

extern "C" void mul_es(int,int,int*);

struct node{
    QByteArray *arr;
    int l;
    node(QByteArray*a, int n){
        l = n;
        arr = a;
    }

};

class t_mult_escalar : public QThread
{   Q_OBJECT
public:
    t_mult_escalar();
    void set_up(int n , int o_cs, int cs, int x_s, int Id);
```

```
protected:
    void run();
signals:
    void finished();
    void done(int*,int);
private:
    void get_array(int* arr, QByteArray* original,int c_s);
    QVector<node*>* pending;
    node* line1;
    int escalar;
    int n_cs;
    int c_s;
    int x;
    bool ending;
    int id;
public slots:
    void onWork(QByteArray arr,int n, int Id);
    void onEnd();

};

#endif // T_MULT_ESCALAR_H
---------------
```