

PRÁCTICA FINAL IMÁGENES

VERSIÓN MATRIZ

Realizado por :

- Víctor Urquiza Hinojosa 15474852S
- Rafael Lachica Garrido 76655442F

ÍNDICE

[VERSIÓN MATRIZ](#)

1. [CLASE IMAGEN](#)
2. [GETTERS / SETTERS](#)
3. [SUMA IMAGEN](#)
4. [ARTEASCII:](#)
5. [LEER / ESCRIBIR IMAGEN](#)
6. [OTRAS PRUEBAS](#)

1. CLASE IMAGEN

Ahora en la clase imagen, la representaci3n interna de la variable datos, ser1 la de una matriz, en la que la ahora cada posici3n n-fila apuntar1 al vector de filas*columnas, de forma que la posici3n 1 apuntar1 a la posici3n del vector de filas*columnas en la posici3n 1-i con un desplazamiento de ncolumnas. Con esto podemos hacer que la posici3n datos[1] apunte la posici3n 00 de la matriz, la posici3n datos[2] apunte a datos[1][3] en la que a su vez datos[1] es igual a datos[0][0], por lo que estar1amos apuntando siempre a datos[0]+desplazamiento, donde el desplazamiento ser1a **i-1*ncolumnas**.

Datos es ahora un puntero de punteros de tipo byte (unsigned char)

```
byte ** datos;
```

M3todo crear que reserva memoria y crea la matriz seg3n lo dicho anteriormente. Se llamar1 en los constructores:

```
void Imagen::crear(int filas,int columnas){
    if (datos!=0) {
        //Si hay memoria reservada
        destruir();
    }
    this->nfilas = f;
    this->ncolumnas = c;

    datos = new byte * [nfilas]; //array de punteros

    datos[0]= new byte [nfilas*ncolumnas]; //Primera columna que apunta al array

    //Enlazamos desde la posici3n 1 al final
    for (int i=1; i< nfilas; i++) {
        datos [i] = datos [i-1] + ncolumnas;
    }

    //asignar valor
    for (int i=0; i<nfilas; i++)
        for (int j=0; j<ncolumnas; j++)
            datos[i][j] = 0;
}
}
```

M3todo destruir que se llama en el destructor:

```
void Imagen::destruir(){
    if (datos != 0){
        if (datos[0]!=0)
            delete [] datos [0]; //Borramos la primera fila
    }
}
```

```

        delete [] datos;
    }
    datos = 0;
    nfilas = ncolumnas = 0;
}

```

Por seguridad comprobamos que el puntero sea distinto de nulo con `datos!=0`

2. GETTERS / SETTERS

Métodos GET/SET:

Para el get vemos como obtenemos la posición como una matriz normal, al haber hecho el enlace en el constructor ahora podemos acceder como a una matriz:

```

void Imagen::set(int y, int x, byte v){
    if (this->nfilas > y && this->ncolumnas > x)
        datos[y][x]=v;
    else {
        cerr<<"Error posicion más allá la matriz en el set"<<endl;
        exit(1);
    }
}

```

Igual que el método anterior, salvo que el objeto que llama a la función es constante, lo usamos para el constructor de copia y asignación:

```

const byte Imagen::get(int y, int x) const {
    if (this->nfilas > y && this->ncolumnas > x)
        return datos[y][x];
    else {
        cerr<<"Error posicion incorrecta más allá de la matriz en el
get"<<endl;
        exit(1);
    }
}

```

Obtenemos una casilla de la matriz como si de un vector se tratase y lo modificamos

```

void Imagen::setPos(int i, byte b){
    //Comprobamos que esta dentro de los limites
    if (i<nfilas*ncolumnas)
        datos[0][i]=b;
    else {

```

```

        cerr<<"Error posicion incorrecta más allá de la matriz "<<endl;
        exit(1);
    }
}

```

Igual forma de obtener la posición que antes

```

byte Imagen::getPos(int i){
    //Comprobamos que esta dentro de los limites
    if (i<nfilas*ncolumnas)
        return datos[0][i];
    else {
        cerr<<"Error posicion incorrecta más allá de la matriz "<<endl;
        exit(1);
    }
}

```

Método para obtener el byte de una posición, donde se llama desde un objeto imagen constante y devuelve un byte constante

```

const byte Imagen::getPos(int i) const {
    //Comprobamos que esta dentro de los limites
    if (i<nfilas*ncolumnas)
        return datos[0][i];
    else {
        cerr<<"Error posicion incorrecta más allá de la matriz "<<endl;
        exit(1);
    }
}

```

3. SUMA IMAGEN

Función Suma que añade en una Imagen final el resultado de sumar la Imagen A y la Imagen B:

```

Imagen operator + (const Imagen imagenA, const Imagen imagenB){
    Imagen nueva;
    if (imagenA.filas()>0 && imagenB.filas()>0) {
        int nuevaFila = imagenA.filas() > imagenB.filas() ? imagenA.filas()
: imagenB.filas(); //Nos quedamos con el mayor numero de filas
        int nuevaColumna = imagenA.columnas()+imagenB.columnas(); //Sumamos
el numero de columnas

        nueva.crear(nuevaFila,nuevaColumna);

        // //Escribimos la imagen A

```

```

for (int i=0; i<imagenA.filas(); i++)
    for (int j=0; j<imagenA.columnas(); j++)
        nueva.setPos(i*nueva.columnas()+j,imagenA.get(i,j));

```

Vemos cómo recorremos y ponemos en la matriz [0][desplazamiento $i \cdot \text{columnas} + j$] el valor de la imagen A

```

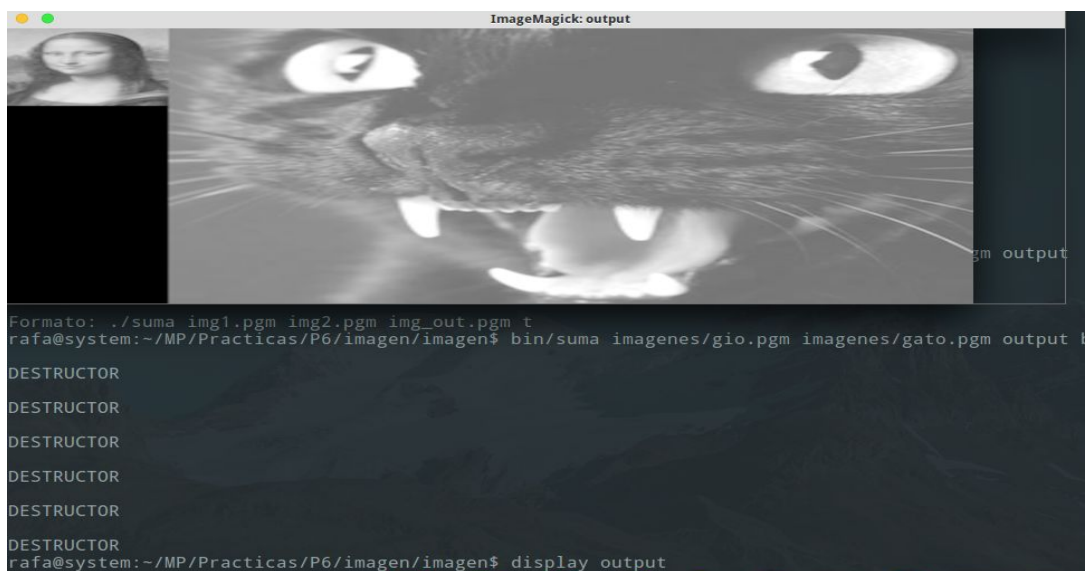
//Escribimos la imagen B
for (int i=0; i<imagenB.nfilas; i++)
    for (int j=0; j<imagenB.ncolumnas; j++)

nueva.setPos(i*nueva.columnas()+j+imagenA.columnas(),imagenB.get(i,j));

}
else{
    cerr<<"\nDevolviendo imagen nula, una de las 2 imagenes no
existe:"<<endl;
}

return nueva;

```



4. ARTEASCII:

Funci3n arteASCII:

Obtenemos el cardinal del array de grises, y calculamos su valor ponderándolo en el píxel fila,columna. Ese valor ascii lo guardamos en la posici3n correspondiente en arteASCII.

```
const bool Imagen::aArteASCII (const char grises[],char ** arteASCII,int maxlong){
int filas = this->nfilas;
    int columnas = this->ncolumnas;
    int cardinal;
    byte pixel;
    bool status=true;
    cardinal = 0;
    while (grises[cardinal] != ' ') { //Recorre la lista hasta que encuentra el
espacio final
        cardinal++;
    }

    //Si es mayor que filas*columnas cabe en la imagen, la creamos y devuelve
true
    if (maxlong > (filas*columnas)) {
        for (int i=0; i<filas; i++) {
            //Obtenemos valor del pixel
            for (int j=0; j<columnas; j++) { //Una columna menos
porque la ultima tiene el caracter \n
                pixel = this->get(i,j);
                //Asignamos a cada posici3n de arteASCII su
conversion a caracteres segun el rango de intensidad
                //de cada pixel
                arteASCII[i][j] = grises[pixel*cardinal/256];
            }
        }

    }

    else{
        status = false;
    }

    return status;
}
```

Si todo va bien, devuelve true

Funci3n listaaArteASCII

De una lista de grises, crea una imagen arteASCII, donde llama al ḿtodo anterior y le pasa un componente de la lista, en este caso un string que es el "gris":

```
const bool Imagen::listaAArteASCII(const Lista &celdas){

//Reserva memoria para el arteASCII

    char ** arteASCII = 0;
    arteASCII = new char * [nfilas]; //array de punteros

    arteASCII[0]= new char [nfilas*ncolumnas]; //Primera columna que apunta al
array

    //Enlazamos desde la posicion 1 al final
    for (int i=1; i< nfilas; i++) {
        arteASCII [i] = arteASCII [i-1] + ncolumnas;
    }

    bool exito=true;
    //asignar valor
    for (int i=0; i<nfilas; i++)
        for (int j=0; j<ncolumnas; j++)
            arteASCII[i][j] = 0;

    //Para cada lista de celdas
    for(int x=0; x<celdas.longitud(); x++) {
        string gris = celdas.getCelda(x);
        const char *gris_char = gris.c_str();

        //strcpy(gris_char, gris.c_str());
        bool estado;

        estado = this->aArteASCII(gris_char, arteASCII,
this->nfilas*(this->ncolumnas+1)+1);

        if(estado) {
            char nombre_aux[255]="";
            ofstream fsalida;
            cout<<nombre_aux<<" "<<"ascii"<<x<<".txt"<<endl;
            sprintf(nombre_aux, "ascii%d.txt",x); //Formateamos el
nombre_aux

            if (arteASCII !=NULL) {
                fsalida.open(nombre_aux);

imprimir(fsalida,arteASCII,this->nfilas,this->ncolumnas);
                fsalida.close();
            }
        }else{

```

```

        cout << "La conversión " << x << " no ha sido posible" <<
endl;

        exito= false;
    }
}

//Liberamos memoria igual que en el destructor
if (arteASCII[0] != 0) //Si fila de filas * columnas no esta vacia
    delete [] arteASCII [0]; //Borramos la primera fila

if (arteASCII != 0)
    delete [] arteASCII;

arteASCII = 0;

return exito;
}

```

```

=====
Imprimiendo lista de grises
Lista de longitud : 4
#$:
@=Q[]?
@=*=--.,
$@%$&vM#*oahkbp\qwmZ00QLCJUYxzcvunxrjft/|{}[]?~_+<+iill;,"A''.
=====
ascii0.txt
ascii1.txt
ascii2.txt
ascii3.txt
Exito en la conversion.
Chequee los ficheros asciiX.txt

DESTRUCTOR
==4301==
==4301== HEAP SUMMARY:
==4301==   in use at exit: 0 bytes in 0 blocks
==4301== total heap usage: 31 allocs, 31 frees, 71,538 bytes allocated
==4301==
==4301== All heap blocks were freed -- no leaks are possible
==4301==
==4301== For counts of detected and suppressed errors, rerun with: -v
==4301== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
rafa@system:~/MP/Practicas/P6/imagen/imagen$

```

5. LEER / ESCRIBIR IMAGEN

Funciones Leer / Escribir Imagen

```
bool Imagen::leerImagen(const char nombreFichero[]){
```



```

        int f, c;
        TipoImagen tipo = infoPGM(nombreFichero,f,c);
        bool status = false;
        if(tipo==IMG_PGM_BINARIO) {
            this->crear(f,c);
leerPGMBinario(nombreFichero,datos,this->nfilas,this->ncolumnas);
            status = true;
        }else{
            if(tipo==IMG_PGM_TEXTO) {
                this->crear(f,c);
leerPGMTexto(nombreFichero,datos,f,c);
                status = true;
            }
        }
        return status;
    }
}

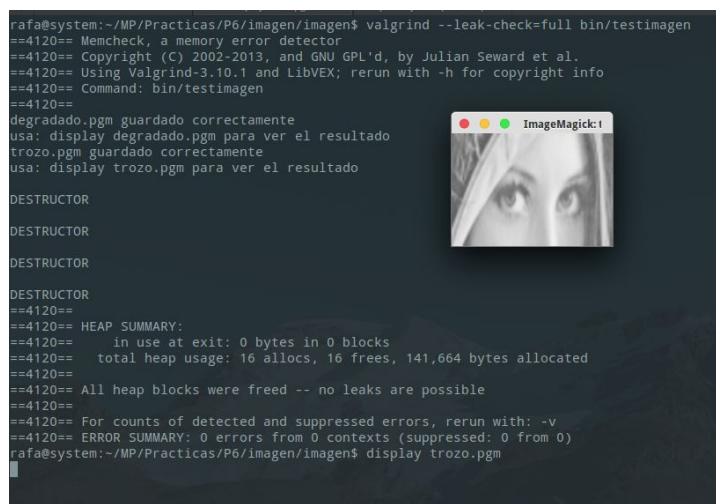
const bool Imagen::escribirImagen(const char nombreFichero[], bool esBinario){
    return (esBinario) ?
        escribirPGMBinario (nombreFichero, (const_cast<const byte
**>(this->datos)), this->nfilas, this->ncolumnas) :
        escribirPGMTexto(nombreFichero, (const_cast<const byte
**>(this->datos)), this->nfilas, this->ncolumnas);
    //Pasamos los valores casting a constante para que no se modifiquen
}

```

Vemos que por seguridad, si la lectura del tipo fué correcta, entonces creamos la matriz de datos.

6. OTRAS PRUEBAS

Testimagen



```

rafa@system:~/MP/Practicas/P6/imagen/imagen$ valgrind --leak-check=full bin/testimagen
==4120== Memcheck, a memory error detector
==4120== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4120== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==4120== Command: bin/testimagen
==4120==
degradado.pgm guardado correctamente
usa: display degradado.pgm para ver el resultado
trozo.pgm guardado correctamente
usa: display trozo.pgm para ver el resultado

DESTRUCTOR
DESTRUCTOR
DESTRUCTOR
DESTRUCTOR
==4120==
==4120== HEAP SUMMARY:
==4120==    in use at exit: 0 bytes in 0 blocks
==4120==   total heap usage: 16 allocs, 16 frees, 141,664 bytes allocated
==4120==
==4120== All heap blocks were freed -- no leaks are possible
==4120==
==4120== For counts of detected and suppressed errors, rerun with: -v
==4120== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
rafa@system:~/MP/Practicas/P6/imagen/imagen$ display trozo.pgm

```

