Índice de Contenido

1. Programa (C++)

El programa está realizado en C++. En él se hace uso de imágenes digitales, en escala de grises. En el programa, primeramente, se extraen planos de bits de una imagen. En la segunda parte se crea arte ASCII que consiste en representar imágenes con los 95 caracteres imprimibles definidos en el estándar ASCII. Para realizar este programa se necesita usar el archivo bloqueLED creado en la practica anterior.

1.1. Funciones Imagen.cpp

1.1.1. Constructores

Hay dos constructores: uno sin parámetros y otro con parámetros donde se le pasan el número de filas y columnas.

Su función es inicializar las filas y columnas. En el constructor con parámetros se inicializa también el vector con los datos.

Funcionamiento:

- En ambos casos se inicializan nfilas y ncolumnas. En el constructor sin parámetros se inicializa a 0. En el constructor con parámetros se inicializa con los valores dados.
- En el constructor con parámetros se inicializa el vector datos[] a 0.

```
Imagen::Imagen(){
        nfilas = 0;
        ncolumnas = 0;
}

Imagen::Imagen(int filas, int columnas){
        nfilas = filas;
        ncolumnas = columnas;

        for(int i = 0; i < nfilas*ncolumnas; i++){
                datos[i] = 0;
        }
}
```

1.1.2. Función crear()

Se le pasan dos enteros con el número de filas y de columnas. Se igualan con nfilas y ncolumnas y se pone el vector datos[] a 0 en todas sus posiciones útiles.

Funcionamiento:

- Se igualan nfilas y ncolumnas con los parámetros dados.
- Se inicializa el vector datos[] en 0.

```
void Imagen::crear(int filas, int columnas){
        nfilas = filas;
        ncolumnas = columnas;

        for(int i = 0; i < nfilas*ncolumnas; i++){
                datos[i] = 0;
        }
}
```

1.1.3. Función filas()

Función que devuelve el número de filas de la imagen.

Funcionamiento:

- Devuelve el entero nfilas.

```
int Imagen::filas(){
        return nfilas;
}
```

1.1.4. Función columnas()

Función que devuelve el número de columnas de la imagen.

Funcionamiento:

- Devuelve el entero ncolumnas.

```
int Imagen::columnas(){
        return ncolumnas;
}
```

1.1.5. Función set()

Se le pasan las filas, columnas y un byte. Se calcula la posición a partir de filas y columnas. En esa posición en datos[] se le inserta el byte dado.

Funcionamiento:

- Se calcula el valor de la posición del byte a modificar.
- Se modifica el valor del byte de esa posición con el valor dado.

```
void Imagen::set(int y, int x, byte v){
        datos[y*ncolumnas+x] = v;
}
```

1.1.6. Función get()

Se le pasan dos enteros con filas y columnas. Se calcula la posición a partir de estas y de vuelve el byte en esa posición.

Funcionamiento:

- Devuelve el estado del byte en la posición dada.

```
byte Imagen::get(int y, int x){
        return datos[y*ncolumnas+x];
}
```

1.1.7. Función setPos()

Recibe un entero con la posición y un byte. En la posición dada, en el vector datos[], se inserta el byte.

Funcionamiento:

- Se pasa la posición y el byte para sustituir el byte en esa posición de datos[] por el dado.

```
void Imagen::setPos(int i, byte v){
        datos[i] = v;
}
```

1.1.8. Función getPos()

Se le pasa un entero con la posición. Devuelve el byte en la posición indicada.

Funcionamiento:

- Se hace un return de datos[] devolviendo el byte en la posición que se le pasa.

```
byte Imagen::getPos(int i){
        return datos[i];
}
```

1.1.9. Función leerImagen()

Se le pasa un array de char con el nombre del fichero. Se comprueba el fichero y se lee cargándolo en memoria. Esto es rellenar nfilas, ncolumnas y datos[] con la información del fichero.

Funcionamiento:

- Se usa un bool en false para controlar que se haga la lectura correctamente.
- Con infoPGM() se comprueba que el fichero es válido.
- Si es válido, se usa leerPGMBinario() para cargar los datos.
- Devuelve el bool que indica si se ha leído correctamente o no.

```
bool Imagen::leerImagen(const char nombreFichero[]){
        bool flag = false;
        TipoImagen tipo = infoPGM(nombreFichero, nfilas, ncolumnas);

        if((tipo == IMG_PGM_BINARIO) && ((nfilas*ncolumnas) < MAXPIXELS)){
                flag = leerPGMBinario(nombreFichero, datos, nfilas, ncolumnas);
        }

        return flag;
}
```

1.1.10. Función escribirImagen()

Se le pasa un array de char con el nombre del fichero y un bool. Devuelve un bool para indicar si se ha escrito correctamente el fichero.

Funcionamiento:

- Se usa la función escribirPGMBinario().

- Los datos a escribir son los que se encuentran en memoria (datos[], nfilas, ncolumnas).
- Devuelve el bool que indica si se ha escrito la imagen.

```
bool Imagen::escribirImagen(const char nombreFichero[], bool esBinario){
        return escribirPGMBinario(nombreFichero, datos, nfilas, ncolumnas);
}
```

1.1.11. Función plano()

Dado un número, k, extrae el plano de bits k- ésimo de la imagen actual y lo devuelva como una nueva imagen.

Funcionamiento:

- Se le pasa el numero k, se crea una imagen de nfilas y ncolumnas y un byte auxiliar.
- Se hace un bucle que recorre todas las posiciones del vector de la imagen.
- Se extra el byte en la posición i de la nueva imagen creada.
- Se comprueba si el bit de datos[i] en la posición k esta encendido o apagado.
- En aux se enciende o apaga el bit en la posición 7 según lo anterior.
- Aux sustituye al byte en la imagen creada.
- Devuelve la imagen nueva.

```
Imagen Imagen::plano(int k){
        Imagen img(nfilas, ncolumnas);
        byte aux;

        for(int i = 0; i < nfilas*ncolumnas; i++){
                aux = img.getPos(i);
                if(getbit(datos[i], k)){
                        on(aux, 7);
                }else{
                        off(aux, 7);
                }
                img.setPos(i, aux);
        }

        return img;
}
```

1.1.12. Función aArteASCII()

Función que transforma una imagen en un conjunto de caracteres ASCII que dibujan la misma imagen dada. Se le pasa el conjunto de caracteres que se usaran para pintar la imagen (grises), el vector donde se guardara la imagen (arteASCII) y el tamaño máximo del vector.

Funcionamiento:

- Primero se cuentan el número de caracteres en grises[] para los posteriores cálculos.
- Se hace un bucle con dos contadores, uno de lectura y uno de escritura puesto que habrá un desfase al introducir los saltos de línea.
- Usando el cardinal de grises[] se calcula a cada byte de datos[] que carácter de grises[] le corresponde y se guarda en arteASCII[].
- Si llega al final de la línea, denotado por la cantidad de columnas, se introduce un salto de línea.
- Al terminar se inserta un salto de línea al final para dejar la imagen ASCII terminada y se devuelve un bool que indica si se ha podido dibujar la imagen completa o ha faltado espacio para hacerlo.

```
bool Imagen::aArteASCII(const char grises[], char arteASCII[], int maxlong){
        int cardinal = 0, contadorRead = 0, aux = 0, contadorWrite = 0;
        bool flag = (nfilas*ncolumnas) < maxlong;

        while (grises[cardinal]!='\0'){
                cardinal++;
        }

        while(contadorRead < nfilas*ncolumnas && contadorWrite < maxlong){
                if(contadorRead%ncolumnas == 0 && contadorRead >= ncolumnas){
                        arteASCII[contadorWrite] = 10;
                        contadorWrite++;
                }
                aux = datos[contadorRead];
                arteASCII[contadorWrite] = grises[aux*cardinal/256];
                contadorRead++;
                contadorWrite++;
        }

        if(contadorWrite+1 < maxlong){
                arteASCII[contadorWrite] = 10;
                arteASCII[contadorWrite+1] = '\0';
        }

        return flag;
}
```

1.2. arteASCII

Se pide construir un programa principal que pida por teclado el nombre de la imagen a codificar y los caracteres que se usaran para dibujarla. Imprime en pantalla el resultado final. En el programa se ha incluido una función para leer los datos desde teclado.

```
const int TOPE = 257;

void leer(char *s){
    do{
        cin.getline(s, TOPE-1);
    }while (s[0]=='\0');
}

int main(){
    char arteASCII[4501];
    char grises[TOPE];
    char nombre[TOPE];
   Imagen origen;

    cout << "Introduce el Nombre y la Ruta de la Imagen de Entrada: ";
    leer(nombre);

    if (!origen.leerImagen(nombre)){
        cerr << "Error en la Lectura de " << nombre << "\n";
        return 1;
    }

    cout << "Introduce el Conjunto de Caracteres: ";
    leer(grises);

    cout << "\nLa imagen en arte ASCII es:\n";
    if(origen.aArteASCII(grises, arteASCII, 4500))
        cout << arteASCII;
    else
        cout << "La conversión no ha sido posible" << endl;
}
```

1.3. Problemas Surgidos

El único problema surgido ha sido con la implementación del bucle en aArteASCII ya que eran necesarios dos contadores, uno de lectura que no se modificara y otro de escritura que incluye los saltos de línea que originalmente no estaban.

2. Makefile

Los makefiles son los ficheros de texto que utiliza make para llevar la gestión de la compilación de programas. Se nos pide crear un fichero makefile con la intención de automatizar el proceso de compilación de nuestro programa.

El uso del fichero makefile se realiza usando el comando make. Las reglas están preparadas para funcionar tanto con un make all como con un make de una regla en concreto.

En primer lugar hemos creado las variables que contienen los nombres de los directorios necesarios.

```
SRC = src
INC = include
OBJ = obj
BIN = bin
LIB = lib

CXX = g++
CPPFLAGS = -Wall -g -c -I$(INC)
```

Una vez hecho esto, hemos creado las reglas necesarias para la compilación del programa.

```
all: $(BIN)/arteASCII $(BIN)/testarteASCII $(BIN)/testplano $(BIN)/testimagen


$(BIN)/testimagen: $(OBJ)/testimagen.o $(LIB)/libimagen.a
	$(CXX) -o $(BIN)/testimagen $(OBJ)/testimagen.o -L$(LIB)/ -limagen

$(BIN)/testplano: $(OBJ)/testplano.o $(LIB)/libimagen.a
	$(CXX) -o $(BIN)/testplano $(OBJ)/testplano.o -L$(LIB)/ -limagen

$(BIN)/testarteASCII: $(OBJ)/testarteASCII.o $(LIB)/libimagen.a
	$(CXX) -o $(BIN)/testarteASCII $(OBJ)/testarteASCII.o -L$(LIB)/ -limagen

$(BIN)/arteASCII: $(OBJ)/arteASCII.o $(LIB)/libimagen.a
	$(CXX) -o $(BIN)/arteASCII $(OBJ)/arteASCII.o -L$(LIB)/ -limagen

$(LIB)/libimagen.a: $(OBJ)/pgm.o $(OBJ)/imagen.o $(OBJ)/byte.o
	ar rsv $(LIB)/libimagen.a $(OBJ)/pgm.o $(OBJ)/imagen.o $(OBJ)/byte.o


$(OBJ)/testimagen.o: $(SRC)/testimagen.cpp
	$(CXX) -c $(SRC)/testimagen.cpp -o $(OBJ)/testimagen.o $(CPPFLAGS)

$(OBJ)/testplano.o: $(SRC)/testplano.cpp
	$(CXX) -c $(SRC)/testplano.cpp -o $(OBJ)/testplano.o $(CPPFLAGS)

$(OBJ)/testarteASCII.o: $(SRC)/testarteASCII.cpp
	$(CXX) -c $(SRC)/testarteASCII.cpp -o $(OBJ)/testarteASCII.o $(CPPFLAGS)

$(OBJ)/arteASCII.o: $(SRC)/arteASCII.cpp
	$(CXX) -c $(SRC)/arteASCII.cpp -o $(OBJ)/arteASCII.o $(CPPFLAGS)


$(OBJ)/imagen.o: $(SRC)/imagen.cpp
	$(CXX) -c $(SRC)/imagen.cpp -o $(OBJ)/imagen.o $(CPPFLAGS)

$(OBJ)/pgm.o: $(SRC)/pgm.cpp
	$(CXX) -c $(SRC)/pgm.cpp -o $(OBJ)/pgm.o $(CPPFLAGS)

$(OBJ)/byte.o: $(SRC)/byte.cpp
	$(CXX) -c $(SRC)/byte.cpp -o $(OBJ)/byte.o $(CPPFLAGS)
```

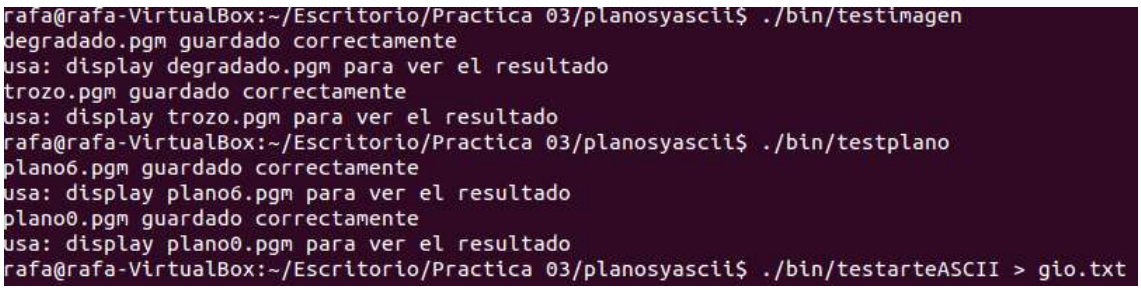
También se incluyen reglas para la limpieza de archivos y para revisar la documentación doxygen.

```
# ************ documentación *******

documentacion:
	doxygen doc/doxys/Doxyfile

# ************ Limpieza ************

clean:
	echo "Limpiando..."
	rm $(OBJ)/*.o $(LIB)/lib*.a

mrproper: clean
	rm $(BIN)/testimagen $(BIN)/testplano $(BIN)/testarteASCII $(BIN)/arteASCII
```

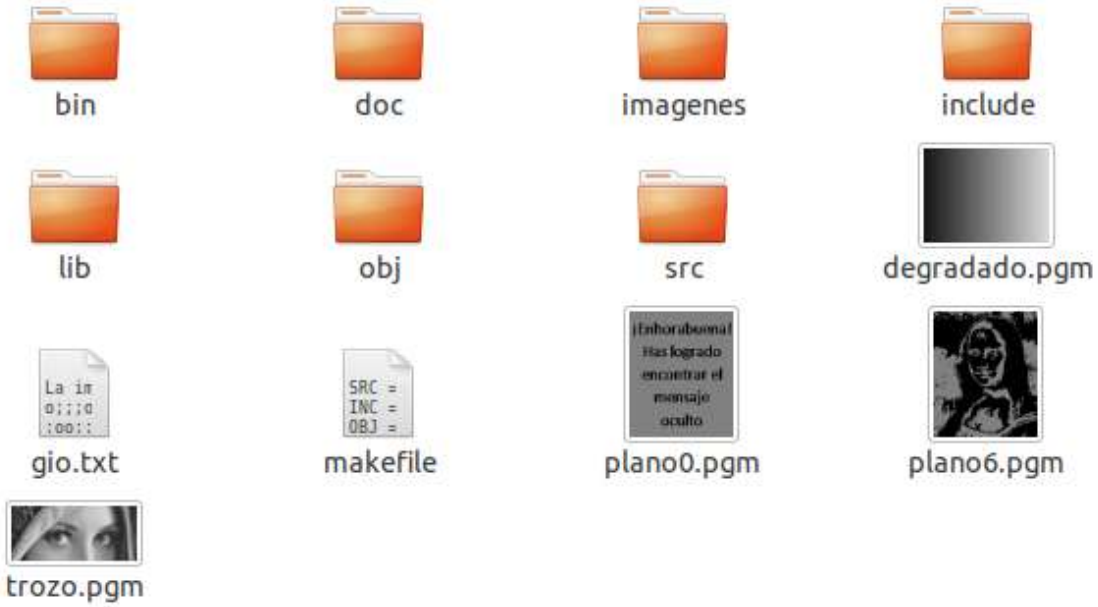
3. Ejecución del Programa

En este apartado se pueden ver imágenes del programa siendo ejecutado en el terminal de Ubuntu. También se adjuntan capturas del uso del fichero makefile para atestiguar que todo funciona correctamente y de acuerdo a las especificaciones dadas.

En esta primera imagen vemos el funcionamiento del archivo makefile y como genera todos los archivos correctamente.

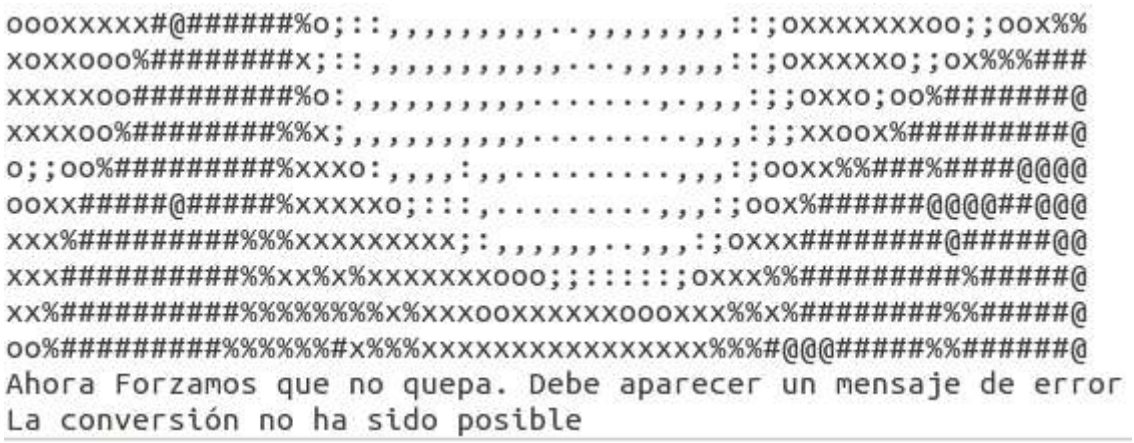


En esta siguiente captura se puede ver la ejecución de los tres archivos test y como realizan sus funciones correctamente. A continuación mostramos como genera los archivos correctamente y con el contenido adecuado.



El contenido de gio.txt se muestra en la siguiente captura para atestiguar que la ejecución del último ejecutable test se realiza correctamente.

```
La imagen en arte ASCII es:
o;;;o;o;ooooooooooo;ooooooooooo;;;;;;ooooooo;;;;;;ooooo;oo
;oo;;;;;;;ooo;ooo;;;;;ooo0;;;;;oo;;;;;;ooooo;;;;;;;;ooo0;;oo
;;;;;;;;;;;;;;;o;;;;;;;o;;;;;;;;;;;;;;;ooooo;;;;;;;ooooooooo
o;;;;;;;;;;;;o;;;;o;;;;o;;;;;oxx%%%%%xoooooooo;;;;;;ooooo;;o
;;;;;;;;;;;;;;;;;;;;;;;;;;;ox%%##########%o;o;;;;;;;;;ooooo;;
;;;;;;;;;;;;;;;;;;;;;;;;;;x%#%x%%##########%o;;;;;;;;oo;;;o;;
;;;;;;;;;;;;;;;;;;;;;;;;;x%%%xx%%%%%#######@@x;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;x%xxxoxx%%%%#####@##@@%;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;x%xxxxxxx%%%%%#####@@#@@%;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;:x%%xxoo;ooxx%%%#####@@@#@@x;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;%%o;;:::::;;;ox%%#####@@@@@@o;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;x%o;:::,,:::;;ox%%#####@@@@@@#;;;;;;;;;;
;;;;;:;;;;;;;;;;;;;;;o%x;:,,,,,,,:::;ox%%###@@@@@@@@x;;;;;;;;;;
:;:;;;;;;;;;;;;;;;;;%#o:,,,,,,,,,:::;ox%###@@@@@@@@#;;;;;;;;;;
:::;;;;;;;;;;;;;;;;x#%;:,,,,,,,,,:::;ox%%####@@@@@@x;;;;;;;;;;
:::::::::;;;;;;;;;;;x#%;:,,,,,,,,,:::;oxx%####@@@@@@#;;;;;;;;;;
:::::::;;;;;;;;;;;;ox#x;:::,:,,,,:::::;ooxx%##@@@@@@@@ooo;;;o;;
:::::::::;;;;;;;;;;;x##x;:::,,,:,,:::::;;ooox%###@@@@@@xoooooo;;
;;;;;;;;;;;;;;;;;;;o%##o;:::,,,,,,:::::;;;;ox%###@@@@@@%oxoooo;;
;;;;;;;;;;;;;;;;;;;x###o;:::::,,,,,:::::;;;;ox%##@@@@@@@#oxxxxooo
;;o;;;;;;;;;;,;;;;x###o:::::::,,:::;;;;;;;;oox%###@@@@@@@#x%xooooo
ooooo;:::::::::ox###o;;;o;:::;oxxooooooxxx%###@@##@@#x%xxxooo
oooxoooo::::::;xx%%#xxoooo;::o%xo;ox%%%%x%###@@@@@@@%%xxxxoo
xxxxoooo::::::;x%%%%%%xx%xo::xxo;x%%%%%oox####@@@@@@%x%%xxxx
xxxxooxo:::;;;;;o%###oo;;xo;,:xo;:;oooo;;ox####@@@#@@%%%%%xxx
xxxxoox;;;oxo;;x#%##;;;;o;;,:oo;;;;o;;:;ox####@@@#@@%%%%xxxx
```



La ultima captura corresponde a la ejecución de arteASCII, el programa principal que se pedía crear donde los parámetros se piden por teclado.