	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Proyecto final:transformación de imágenes

Profesor(a): Oscar René Valdez Casillas

Asignatura: Fundamentos de Programación

Grupo: 21

No de Práctica(s): 12

Integrante(s): ***Santiago Durán Rendón***

Santiago Noriega Chiu

Jesús Ramírez Reyes

No. de lista o brigada: 01

Semestre: 2025-1

Fecha de entrega: 20/11/2024

Observaciones:

CALIFICACIÓN: _____

Brigada 1. (2024). *Proyecto final: Transformación de imágenes*. UNAM

https://github.com/SARD82/PROYECTO-FINAL.git

Índice

NOTA: Para visualizar de mejor manera el código (FINAL_p) y el pseudocódigo (pseudocodigo) entrar al link del github.

Pseudocódigo

```
INICIO Proyecto_final


  // Definición de constantes

  DEFINIR MAX_ANCHO = 512

  DEFINIR MAX_ALTO = 512


  // Declaración de variables enteras

  ENTERO alto, ancho, umbral

  ENTERO alto_frente, ancho_frente, alto_fondo, ancho_fondo


  // Declaración de matrices

  MATRIZ matriz[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER

  MATRIZ canal_rojo[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER

  MATRIZ canal_verde[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER

  MATRIZ canal_azul[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER

  MATRIZ grises[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER

  MATRIZ bn[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER
```

```
// Declaración de arreglos para histogramas
ARREGLO histR[256] TIPO ENTERO
ARREGLO histG[256] TIPO ENTERO
ARREGLO histB[256] TIPO ENTERO
ARREGLO histGris[256] TIPO ENTERO

// Matrices para procesamiento de imágenes
MATRIZ frente[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER
MATRIZ fondo[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER
MATRIZ mezcla[MAX_ALTO][MAX_ANCHO][3] TIPO CARACTER

FUNCION PRINCIPAL(argumentos, valores[])

  // Verificación inicial de argumentos
  SI (argumentos < 2 O argumentos > 6) ENTONCES
    IMPRIMIR "Error: No se proporcionaron los argumentos correctos."
    IMPRIMIR "Use el comando -help para obtener mas informacion."
  FIN SI

  // Verificar si se solicita ayuda
  SI (valores[1] = "-help") ENTONCES
    LLAMAR mostrar_ayuda()
```

```
FIN SI


// Leer cabeceras según formato
SI (valores[1] = "1b" O "2b" O "3b" O "4b" O "5b" O "6b" O "7b" O "8b" O "9b") ENTONCES
   LLAMAR leer_cabecera_bmp(valores[2])
SINO SI (valores[1] = "1p" O "2p" O "3p" O "4p" O "5p" O "6p" O "7p" O "8p" O "9p") ENTONCES
   LLAMAR leer_cabecera_pnm(valores[2])
FIN SI


// Procesamiento de comandos BMP
SEGUN valores[1] HACER
   CASO "1b":
      LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
      SEPARAR_CANAL_ROJO(matriz, canal_rojo, alto, ancho)
      SEPARAR_CANAL_VERDE(matriz, canal_verde, alto, ancho)
      SEPARAR_CANAL_AZUL(matriz, canal_azul, alto, ancho)
      ESCRIBIR_IMAGEN("rojo.bmp", canal_rojo)
      ESCRIBIR_IMAGEN("verde.bmp", canal_verde)
      ESCRIBIR_IMAGEN("azul.bmp", canal_azul)
      IMPRIMIR "Se han creado los archivos rojo.bmp, verde.bmp y azul.bmp."
   FIN CASO
```

```
CASO "2b":
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
    SEPARAR_CANAL_ROJO(matriz, canal_rojo, alto, ancho)
    ESCRIBIR_IMAGEN("rojo.bmp", canal_rojo)
    IMPRIMIR "Se ha creado el archivo rojo.bmp."
FIN CASO

CASO "3b":
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
    SEPARAR_CANAL_VERDE(matriz, canal_verde, alto, ancho)
    ESCRIBIR_IMAGEN("verde.bmp", canal_verde)
    IMPRIMIR "Se ha creado el archivo verde.bmp."
FIN CASO

CASO "4b":
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
    SEPARAR_CANAL_AZUL(matriz, canal_azul, alto, ancho)
    ESCRIBIR_IMAGEN("azul.bmp", canal_azul)
    IMPRIMIR "Se ha creado el archivo azul.bmp."
FIN CASO

CASO "5b":
```

```
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
    CONVERTIR_A_GRISES(matriz, grises, alto, ancho)
    ESCRIBIR_IMAGEN("grises.bmp", grises)
    IMPRIMIR "Se ha creado el archivo grises.bmp."
FIN CASO

CASO "5b":
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
    CONVERTIR_A_GRISES(matriz, grises, alto, ancho)
    ESCRIBIR_IMAGEN("grises.bmp", grises)
    IMPRIMIR "Se ha creado el archivo grises.bmp."
FIN CASO

CASO "6b":
    umbral = CONVERTIR_A_ENTERO(valores[3])
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)
    CONVERTIR_A_BN(matriz, bn, alto, ancho, umbral)
    ESCRIBIR_IMAGEN("bn.bmp", bn)
    IMPRIMIR "Se ha creado el archivo bn.bmp."
FIN CASO

CASO "7b":
```

```
    LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)

    // Calcular histogramas
    CALCULAR_HISTOGRAMA_COLOR(matriz, alto, ancho, histR, histG, histB)
    CALCULAR_HISTOGRAMA_GRISES(matriz, alto, ancho, histGris)

    // Escribir archivos
    ESCRIBIR_HISTOGRAMA("histR.txt", histR)
    ESCRIBIR_HISTOGRAMA("histG.txt", histG)
    ESCRIBIR_HISTOGRAMA("histB.txt", histB)
    ESCRIBIR_HISTOGRAMA("histGris.txt", histGris)

    IMPRIMIR "Histogramas generados exitosamente"
FIN CASO

CASO "8b":
    // Verificar argumentos
    SI (argumentos != 5) ENTONCES
        IMPRIMIR "Error: Uso correcto: programa 8b frente.bmp fondo.bmp alpha"
    FIN SI

    // Convertir alpha a entero
```

```
    alpha = CONVERTIR_A_ENTERO(valores[4])

    // Leer imágenes
    LEER_IMAGEN_BMP(valores[2], frente, alto_frente, ancho_frente)
    LEER_IMAGEN_BMP(valores[3], fondo, alto_fondo, ancho_fondo)

    // Verificar dimensiones
    SI (alto_frente != alto_fondo O ancho_frente != ancho_fondo) ENTONCES
        IMPRIMIR "Error: Las imagenes deben tener el mismo tamano"
    FIN SI

    MEZCLAR_IMAGENES(frente, fondo, mezcla, alto_frente, ancho_frente, alpha)
    ESCRIBIR_IMAGEN_BMP("mezcla.bmp", mezcla, alto_frente, ancho_frente)
    IMPRIMIR "Mezcla realizada con exito. Resultado guardado en 'mezcla.bmp'"
FIN CASO

CASO "9b":
    // Verificar argumentos
    SI (argumentos != 6) ENTONCES
        IMPRIMIR "Error: Uso correcto: programa 9b imagen.bmp fondo.bmp umbral alpha"
    FIN SI
```

```
umbral = CONVERTIR_A_ENTERO(valores[4])

alpha = CONVERTIR_A_ENTERO(valores[5])


// Verificar rangos

SI (umbral < 0 O umbral > 255 O alpha < 0 O alpha > 255) ENTONCES

    IMPRIMIR "Error: umbral y alpha deben estar entre 0 y 255"

FIN SI


IMPRIMIR "=== Procesando todos los comandos para BMP ==="


// 1. Separar canales RGB

IMPRIMIR "1. Separando canales RGB..."

LEER_IMAGEN_BMP(valores[2], matriz, alto, ancho)

SEPARAR_CANAL_ROJO(matriz, canal_rojo, alto, ancho)

SEPARAR_CANAL_VERDE(matriz, canal_verde, alto, ancho)

SEPARAR_CANAL_AZUL(matriz, canal_azul, alto, ancho)

ESCRIBIR_IMAGEN("rojo.bmp", canal_rojo)

ESCRIBIR_IMAGEN("verde.bmp", canal_verde)

ESCRIBIR_IMAGEN("azul.bmp", canal_azul)


// 2. Escala de grises

IMPRIMIR "2. Generando imagen en escala de grises..."
```

```
CONVERTIR_A_GRISES(matriz, grises, alto, ancho)

ESCRIBIR_IMAGEN("grises.bmp", grises)


// 3. Blanco y negro

IMPRIMIR "3. Generando imagen en blanco y negro..."

CONVERTIR_A_BN(matriz, bn, alto, ancho, umbral)

ESCRIBIR_IMAGEN("bn.bmp", bn)


// 4. Histogramas

IMPRIMIR "4. Calculando histogramas..."

CALCULAR_HISTOGRAMA_COLOR(matriz, alto, ancho, histR, histG, histB)

CALCULAR_HISTOGRAMA_GRISES(matriz, alto, ancho, histGris)

ESCRIBIR_HISTOGRAMA("histR.txt", histR)

ESCRIBIR_HISTOGRAMA("histG.txt", histG)

ESCRIBIR_HISTOGRAMA("histB.txt", histB)

ESCRIBIR_HISTOGRAMA("histGris.txt", histGris)


// 5. Mezcla de imágenes

IMPRIMIR "5. Mezclando imagenes..."

LEER_IMAGEN_BMP(valores[3], fondo, alto_fondo, ancho_fondo)

SI (alto != alto_fondo O ancho != ancho_fondo) ENTONCES

    IMPRIMIR "Error: Las imagenes deben tener el mismo tamano"
```

```
    FIN SI

    MEZCLAR_IMAGENES(matriz, fondo, mezcla, alto, ancho, alpha)

    ESCRIBIR_IMAGEN("mezcla.bmp", mezcla)


    IMPRIMIR "=== Procesamiento completado ==="

    IMPRIMIR "Archivos generados:"

    IMPRIMIR "- rojo.bmp, verde.bmp, azul.bmp"

    IMPRIMIR "- grises.bmp"

    IMPRIMIR "- bn.bmp"

    IMPRIMIR "- histR.txt, histG.txt, histB.txt, histGris.txt"

    IMPRIMIR "- mezcla.bmp"

FIN CASO


CASO "1p":

    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)

    SEPARAR_CANAL_ROJO(matriz, canal_rojo, alto, ancho)

    SEPARAR_CANAL_VERDE(matriz, canal_verde, alto, ancho)

    SEPARAR_CANAL_AZUL(matriz, canal_azul, alto, ancho)

    ESCRIBIR_IMAGEN("rojo.pnm", canal_rojo)

    ESCRIBIR_IMAGEN("verde.pnm", canal_verde)

    ESCRIBIR_IMAGEN("azul.pnm", canal_azul)

    IMPRIMIR "Se han creado los archivos rojo.pnm, verde.pnm y azul.pnm."
```

```
FIN CASO

CASO "2p":
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
    SEPARAR_CANAL_ROJO(matriz, canal_rojo, alto, ancho)
    ESCRIBIR_IMAGEN("rojo.pnm", canal_rojo)
    IMPRIMIR "Se ha creado el archivo rojo.pnm."
FIN CASO

CASO "3p":
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
    SEPARAR_CANAL_VERDE(matriz, canal_verde, alto, ancho)
    ESCRIBIR_IMAGEN("verde.pnm", canal_verde)
    IMPRIMIR "Se ha creado el archivo verde.pnm."
FIN CASO

CASO "4p":
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
    SEPARAR_CANAL_AZUL(matriz, canal_azul, alto, ancho)
    ESCRIBIR_IMAGEN("azul.pnm", canal_azul)
    IMPRIMIR "Se ha creado el archivo azul.pnm."
FIN CASO
```

```
CASO "5p":
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
    CONVERTIR_A_GRISES(matriz, grises, alto, ancho)
    ESCRIBIR_IMAGEN("grises.pnm", grises)
    IMPRIMIR "Se ha creado el archivo grises.pnm."
FIN CASO

CASO "6p":
    umbral = CONVERTIR_A_ENTERO(valores[3])
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
    CONVERTIR_A_BN(matriz, bn, alto, ancho, umbral)
    ESCRIBIR_IMAGEN("bn.pnm", bn)
    IMPRIMIR "Se ha creado el archivo bn.pnm."
FIN CASO

CASO "7p":
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
    CALCULAR_HISTOGRAMA_COLOR(matriz, alto, ancho, histR, histG, histB)
    CALCULAR_HISTOGRAMA_GRISES(matriz, alto, ancho, histGris)
    ESCRIBIR_HISTOGRAMA("histR.txt", histR)
    ESCRIBIR_HISTOGRAMA("histG.txt", histG)
```

```
    ESCRIBIR_HISTOGRAMA("histB.txt", histB)
    ESCRIBIR_HISTOGRAMA("histGris.txt", histGris)
    IMPRIMIR "Histogramas generados exitosamente"
FIN CASO

CASO "8p":
    SI (argumentos != 5) ENTONCES
        IMPRIMIR "Error: Uso correcto: programa 8p frente.pnm fondo.pnm alpha"
    FIN SI

alpha = CONVERTIR_A_ENTERO(valores[4])
LEER_IMAGEN_PNM(valores[2], frente, alto_frente, ancho_frente)
LEER_IMAGEN_PNM(valores[3], fondo, alto_fondo, ancho_fondo)

    SI (alto_frente != alto_fondo O ancho_frente != ancho_fondo) ENTONCES
        IMPRIMIR "Error: Las imagenes deben tener el mismo tamano"
    FIN SI

    MEZCLAR_IMAGENES(frente, fondo, mezcla, alto_frente, ancho_frente, alpha)
    ESCRIBIR_IMAGEN("mezcla.pnm", mezcla)
    IMPRIMIR "Mezcla realizada con exito. Resultado guardado en 'mezcla.pnm'"
FIN CASO
```

```
CASO "9p":
    SI (argumentos != 6) ENTONCES
        IMPRIMIR "Error: Uso correcto: programa 9p imagen.pnm fondo.pnm umbral alpha"
    FIN SI

    umbral = CONVERTIR_A_ENTERO(valores[4])
    alpha = CONVERTIR_A_ENTERO(valores[5])

    //Verificar dimensiones
SI (alto > MAX_ALTO) O (ancho > MAX_ANCHO) O (alto_fondo > MAX_ALTO) O (ancho_fondo > MAX_ANCHO) ENTONCES
        IMPRIMIR "Error: Las dimensiones exceden los límites permitidos (MAXIMO_ALTOxMAXIMO_ANCHO)"
    FIN SI

    SI (umbral < 0 O umbral > 255 O alpha < 0 O alpha > 255) ENTONCES
        IMPRIMIR "Error: umbral y alpha deben estar entre 0 y 255"
    FIN SI

    IMPRIMIR "=== Procesando todos los comandos para PNM ==="

    // 1. Separar canales RGB
    IMPRIMIR "1. Separando canales RGB..."
    LEER_IMAGEN_PNM(valores[2], matriz, alto, ancho)
```

```
SEPARAR_CANAL_ROJO(matriz, canal_rojo, alto, ancho)

SEPARAR_CANAL_VERDE(matriz, canal_verde, alto, ancho)

SEPARAR_CANAL_AZUL(matriz, canal_azul, alto, ancho)

ESCRIBIR_IMAGEN("rojo.pnm", canal_rojo)

ESCRIBIR_IMAGEN("verde.pnm", canal_verde)

ESCRIBIR_IMAGEN("azul.pnm", canal_azul)


// 2. Escala de grises

IMPRIMIR "2. Generando imagen en escala de grises..."

CONVERTIR_A_GRISES(matriz, grises, alto, ancho)

ESCRIBIR_IMAGEN("grises.pnm", grises)


// 3. Blanco y negro

IMPRIMIR "3. Generando imagen en blanco y negro..."

CONVERTIR_A_BN(matriz, bn, alto, ancho, umbral)

ESCRIBIR_IMAGEN("bn.pnm", bn)


// 4. Histogramas

IMPRIMIR "4. Calculando histogramas..."

CALCULAR_HISTOGRAMA_COLOR(matriz, alto, ancho, histR, histG, histB)

CALCULAR_HISTOGRAMA_GRISES(matriz, alto, ancho, histGris)

ESCRIBIR_HISTOGRAMA("histR.txt", histR)
```

```
    ESCRIBIR_HISTOGRAMA("histG.txt", histG)
    ESCRIBIR_HISTOGRAMA("histB.txt", histB)
    ESCRIBIR_HISTOGRAMA("histGris.txt", histGris)

    // 5. Mezcla de imágenes
    IMPRIMIR "5. Mezclando imagenes..."
    LEER_IMAGEN_PNM(valores[3], fondo, alto_fondo, ancho_fondo)
    SI (alto != alto_fondo O ancho != ancho_fondo) ENTONCES
        IMPRIMIR "Error: Las imagenes deben tener el mismo tamano"
    FIN SI
    MEZCLAR_IMAGENES(matriz, fondo, mezcla, alto, ancho, alpha)
    ESCRIBIR_IMAGEN("mezcla.pnm", mezcla)

    IMPRIMIR "=== Procesamiento completado ==="
    IMPRIMIR "Archivos generados:"
    IMPRIMIR "- rojo.pnm, verde.pnm, azul.pnm"
    IMPRIMIR "- grises.pnm"
    IMPRIMIR "- bn.pnm"
    IMPRIMIR "- histR.txt, histG.txt, histB.txt, histGris.txt"
    IMPRIMIR "- mezcla.pnm"
FIN CASO
```

```
        OTRO:
            IMPRIMIR "Error: Comando no reconocido."
            IMPRIMIR "Use el comando -help para obtener mas informacion."
        FIN OTRO
    FIN SEGUN
FIN FUNCION

FUNCION mostrar_ayuda()
    IMPRIMIR "-------------------------------------------------------------------------------------------------------------"

    IMPRIMIR "Uso del programa:"
    IMPRIMIR "1p o 1b -> Extrar matrices R, G, B de una imagen."
    IMPRIMIR "2p o 2b -> Extraer unicamente la matriz R."
    IMPRIMIR "3p o 3b -> Extraer unicamente la matriz G."
    IMPRIMIR "4p o 4b -> Extraer unicamente la matriz B."
    IMPRIMIR "5p o 5b -> Generar imagen en escala de grises."
    IMPRIMIR "6p umbral o 6b umbral -> Generar imagen en blanco y negro con umbral (0-255)."
    IMPRIMIR "7p o 7b -> Calcular y guardar el histograma."
    IMPRIMIR "8p alpha o 8b alpha -> Mezclar dos imagenes con coeficiente alpha (0-255)."
    IMPRIMIR "9p umbral alpha o 9b umbral alpha -> Realizar todos los procesos sobre las imagenes."
    IMPRIMIR "-help -> Mostrar este menu de ayuda."
```

```
    IMPRIMIR "-----------------------------------------------------------------------------------------------------------"

    IMPRIMIR "Uso: <tipo imagen> <nombre archivo> [parametros adicionales]"
    IMPRIMIR "Ejemplos de uso:"
    IMPRIMIR "programa 1p/1b imagen.pnm"
    IMPRIMIR "programa 2p/2b imagen.pnm"
    IMPRIMIR "programa 3p/3b imagen.pnm"
    IMPRIMIR "programa 4p/4b imagen.pnm"
    IMPRIMIR "programa 5b/5p imagen.bmp"
    IMPRIMIR "programa 6b/6p imagen.bmp umbral"
    IMPRIMIR "programa 7b/7p imagen.bmp"
    IMPRIMIR "programa 8b/8p frente.pnm fondo.pnm alpha"
     IMPRIMIR "programa 9b/9p imagen1.pnm(Esta primera sera la imagen que haga todos los procesos) imagen2.bmp
umbral alpha"

    IMPRIMIR "-----------------------------------------------------------------------------------------------------------"
  FIN FUNCION

  FUNCION leer_cabecera_bmp(nombre_archivo: TEXTO CONSTANTE)
    // Abrir archivo en modo binario
    archivo = ABRIR_ARCHIVO(nombre_archivo, "rb")

    SI (archivo ES NULO) ENTONCES
```

```
    IMPRIMIR "Error: No se pudo abrir el archivo ", nombre_archivo
    RETORNAR
FIN SI

// Declarar variables para la cabecera
DEFINIR tipo, planos, bpp COMO ENTERO_CORTO_SIN_SIGNO
DEFINIR tamano, reservado, offset, tamanoDIB COMO ENTERO_SIN_SIGNO
DEFINIR ancho, alto COMO ENTERO

// Leer campos de la cabecera
LEER_BINARIO(archivo, tipo)
LEER_BINARIO(archivo, tamano)
LEER_BINARIO(archivo, reservado)
LEER_BINARIO(archivo, offset)
LEER_BINARIO(archivo, tamanoDIB)
LEER_BINARIO(archivo, ancho)
LEER_BINARIO(archivo, alto)
LEER_BINARIO(archivo, planos)
LEER_BINARIO(archivo, bpp)

// Mostrar información
IMPRIMIR "----------------------------------------------------------------------------------------------------"
```

```
    IMPRIMIR "Informacion de la cabecera bmp:"
    IMPRIMIR "Tamano del archivo: ", tamano, " bytes."
    IMPRIMIR "Dimensiones: ", ancho, "x", alto, " pixeles."
    IMPRIMIR "Bits por pixel: ", bpp
    IMPRIMIR "Offset de datos: ", offset, " bytes."
    IMPRIMIR "-------------------------------------------------------------------------------------------------------------"

    CERRAR_ARCHIVO(archivo)
FIN FUNCION

FUNCION leer_cabecera_pnm(nombre_archivo: TEXTO CONSTANTE)
    // Abrir archivo en modo lectura
    archivo = ABRIR_ARCHIVO(nombre_archivo, "r")

    SI (archivo ES NULO) ENTONCES
        IMPRIMIR "Error: No se pudo abrir el archivo ", nombre_archivo
        RETORNAR
    FIN SI

    // Declarar variables para la cabecera
    DEFINIR numero_magico[3] COMO CARACTER
    DEFINIR ancho, alto, maxval COMO ENTERO
```

```
// Leer número mágico (primeros dos caracteres)
LEER_FORMATO(archivo, "%2s", numero_magico)

// Procesar comentarios
caracter = LEER_CARACTER(archivo)

MIENTRAS (caracter = '#') HACER
    MIENTRAS (LEER_CARACTER(archivo) != '\n') HACER
        // Continuar leyendo hasta fin de línea
    FIN MIENTRAS
    caracter = LEER_CARACTER(archivo)
FIN MIENTRAS

DEVOLVER_CARACTER(caracter, archivo)

// Leer dimensiones y valor máximo
LEER_FORMATO(archivo, "%d %d %d", ancho, alto, maxval)

// Mostrar información
IMPRIMIR "-----------------------------------------------------------------------------------------------------------"
IMPRIMIR "Informacion de la cabecera pnm:"
```

```
    IMPRIMIR "Numero magico: ", numero_magico
    IMPRIMIR "Dimensiones: ", ancho, "x", alto, " pixeles."
    IMPRIMIR "Valor maximo: ", maxval
    IMPRIMIR "-----------------------------------------------------------------------------------------------------------"

    CERRAR_ARCHIVO(archivo)
  FIN FUNCION

    FUNCION leer_imagen_bmp(nombre_archivo: TEXTO, matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,
*alto: ENTERO, *ancho: ENTERO)
    // Abrir archivo en modo binario
    archivo = ABRIR_ARCHIVO(nombre_archivo, "rb")

    // Declarar variables locales
    DEFINIR bytes_por_linea, padding, i, j COMO ENTERO
    DEFINIR cabecera_bmp[54] COMO CARACTER_SIN_SIGNO

    SI (archivo ES NULO) ENTONCES
      IMPRIMIR "Error al abrir el archivo ", nombre_archivo
      RETORNAR
    FIN SI

    // Leer cabecera BMP
```

```
LEER_BINARIO(archivo, cabecera_bmp, 54)

// Obtener dimensiones
*ancho = CONVERTIR_A_ENTERO(cabecera_bmp[18])
*alto = CONVERTIR_A_ENTERO(cabecera_bmp[22])

// Verificar dimensiones máximas
SI (*ancho > MAX_ANCHO O *alto > MAX_ALTO) ENTONCES
   IMPRIMIR "Error: La imagen excede los limites permitidos (", MAX_ANCHO, "x", MAX_ALTO, ")"
   CERRAR_ARCHIVO(archivo)
   RETORNAR
FIN SI

// Calcular padding
bytes_por_linea = (*ancho) * 3
padding = (4 - (bytes_por_linea MOD 4)) MOD 4

// Leer datos de la imagen
PARA i DESDE (*alto - 1) HASTA 0 DECREMENTO 1 HACER
   PARA j DESDE 0 HASTA (*ancho - 1) HACER
      // Leer componentes BGR
      LEER_BINARIO(archivo, matriz[i][j][2], 1)  // Azul
```

```
            LEER_BINARIO(archivo, matriz[i][j][1], 1)  // Verde
            LEER_BINARIO(archivo, matriz[i][j][0], 1)  // Rojo
        FIN PARA

        // Saltar padding
        MOVER_PUNTERO_ARCHIVO(archivo, padding, ACTUAL)
    FIN PARA

    CERRAR_ARCHIVO(archivo)
FIN FUNCION

FUNCION escribir_imagen_bmp(nombre_archivo: TEXTO, matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)
    // Abrir archivo en modo binario para escritura
    archivo = ABRIR_ARCHIVO(nombre_archivo, "wb")

    // Declarar variables
    DEFINIR cabecera_bmp[54] COMO CARACTER_SIN_SIGNO
    DEFINIR bytes_por_linea, padding, tamano_archivo, i, j COMO ENTERO
    DEFINIR byte_padding COMO CARACTER_SIN_SIGNO = 0

    SI (archivo ES NULO) ENTONCES
        IMPRIMIR "Error al crear el archivo ", nombre_archivo
```

```
    RETORNAR

FIN SI


// Calcular padding y tamaño del archivo

bytes_por_linea = ancho * 3

padding = (4 - (bytes_por_linea MOD 4)) MOD 4

tamano_archivo = 54 + (bytes_por_linea + padding) * alto


// Preparar cabecera BMP

INICIALIZAR_MEMORIA(cabecera_bmp, 0, 54)  // equivalente a memset


// Configurar campos de la cabecera

cabecera_bmp[0] = 'B'

cabecera_bmp[1] = 'M'

ASIGNAR_ENTERO(cabecera_bmp[2], tamano_archivo)

ASIGNAR_ENTERO(cabecera_bmp[10], 54)

ASIGNAR_ENTERO(cabecera_bmp[14], 40)

ASIGNAR_ENTERO(cabecera_bmp[18], ancho)

ASIGNAR_ENTERO(cabecera_bmp[22], alto)

ASIGNAR_CORTO(cabecera_bmp[26], 1)

ASIGNAR_CORTO(cabecera_bmp[28], 24)

ASIGNAR_ENTERO(cabecera_bmp[34], (bytes_por_linea + padding) * alto)
```

```
    // Escribir cabecera
    ESCRIBIR_BINARIO(archivo, cabecera_bmp, 54)

    // Escribir datos de la imagen
    PARA i DESDE (alto - 1) HASTA 0 DECREMENTO 1 HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            // Escribir en orden BGR
            ESCRIBIR_BINARIO(archivo, matriz[i][j][2], 1)  // Azul
            ESCRIBIR_BINARIO(archivo, matriz[i][j][1], 1)  // Verde
            ESCRIBIR_BINARIO(archivo, matriz[i][j][0], 1)  // Rojo
        FIN PARA

        // Escribir padding
        PARA j DESDE 0 HASTA (padding - 1) HACER
            ESCRIBIR_BINARIO(archivo, byte_padding, 1)
        FIN PARA
    FIN PARA

    CERRAR_ARCHIVO(archivo)
FIN FUNCION
```

```
FUNCION separar_canal_rojo_bmp(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, rojo[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    DEFINIR i, j COMO ENTERO

    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            rojo[i][j][0] = matriz[i][j][0]  // Mantener componente rojo
            rojo[i][j][1] = 0                // Eliminar componente verde
            rojo[i][j][2] = 0                // Eliminar componente azul
        FIN PARA
    FIN PARA
FIN FUNCION

FUNCION separar_canal_verde_bmp(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, verde[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    DEFINIR i, j COMO ENTERO

    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            verde[i][j][0] = 0               // Eliminar componente rojo
            verde[i][j][1] = matriz[i][j][1]   // Mantener componente verde
```

```
            verde[i][j][2] = 0              // Eliminar componente azul
        FIN PARA
    FIN PARA
FIN FUNCION

                FUNCION separar_canal_azul_bmp(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,
azul[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    DEFINIR i, j COMO ENTERO

    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            azul[i][j][0] = 0              // Eliminar componente rojo
            azul[i][j][1] = 0              // Eliminar componente verde
            azul[i][j][2] = matriz[i][j][2]   // Mantener componente azul
        FIN PARA
    FIN PARA
FIN FUNCION

    FUNCION leer_imagen_pnm(nombre_archivo: TEXTO, matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,
*alto: ENTERO, *ancho: ENTERO)

    // Declarar variables
```

```
DEFINIR linea[100] COMO CARACTER
DEFINIR max_valor, i, j COMO ENTERO

// Abrir archivo en modo binario
archivo = ABRIR_ARCHIVO(nombre_archivo, "rb")

SI (archivo ES NULO) ENTONCES
   IMPRIMIR "Error al abrir el archivo ", nombre_archivo
FIN SI

// Verificar formato P6
LEER_LINEA(archivo, linea)
SI (linea[0] != 'P' O linea[1] != '6') ENTONCES
   IMPRIMIR "Formato de archivo incorrecto"
   CERRAR_ARCHIVO(archivo)
FIN SI

// Saltar comentarios
REPETIR
   LEER_LINEA(archivo, linea)
MIENTRAS (linea[0] = '#')
```

```
    // Leer dimensiones
    CONVERTIR_TEXTO_A_NUMEROS(linea, ancho, alto)

    // Leer valor máximo
    LEER_LINEA(archivo, linea)
    CONVERTIR_TEXTO_A_NUMERO(linea, max_valor)

    // Leer datos de la imagen
    PARA i DESDE 0 HASTA (*alto - 1) HACER
      PARA j DESDE 0 HASTA (*ancho - 1) HACER
        LEER_BINARIO(archivo, matriz[i][j][0], 1)  // Rojo
        LEER_BINARIO(archivo, matriz[i][j][1], 1)  // Verde
        LEER_BINARIO(archivo, matriz[i][j][2], 1)  // Azul
      FIN PARA
    FIN PARA

    CERRAR_ARCHIVO(archivo)
  FIN FUNCION

        FUNCION escribir_imagen_pnm(nombre_archivo: TEXTO, matriz[MAX_ALTO][MAX_ANCHO][3]:
CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    // Declarar variables
```

```
DEFINIR i, j COMO ENTERO


// Abrir archivo en modo binario para escritura
archivo = ABRIR_ARCHIVO(nombre_archivo, "wb")


SI (archivo ES NULO) ENTONCES
   IMPRIMIR "Error al crear el archivo ", nombre_archivo
FIN SI


// Escribir cabecera PNM
ESCRIBIR_FORMATO(archivo, "P6\n")              // Identificador del formato
ESCRIBIR_FORMATO(archivo, "%d %d\n", ancho, alto)  // Dimensiones
ESCRIBIR_FORMATO(archivo, "255\n")             // Valor máximo


// Escribir datos de la imagen
PARA i DESDE 0 HASTA (alto - 1) HACER
   PARA j DESDE 0 HASTA (ancho - 1) HACER
      ESCRIBIR_BINARIO(archivo, matriz[i][j][0], 1)  // Rojo
      ESCRIBIR_BINARIO(archivo, matriz[i][j][1], 1)  // Verde
      ESCRIBIR_BINARIO(archivo, matriz[i][j][2], 1)  // Azul
   FIN PARA
FIN PARA
```

```
    CERRAR_ARCHIVO(archivo)
  FIN FUNCION

                FUNCION separar_canal_rojo_pnm(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, rojo[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    DEFINIR i, j COMO ENTERO

    PARA i DESDE 0 HASTA (alto - 1) HACER
      PARA j DESDE 0 HASTA (ancho - 1) HACER
        rojo[i][j][0] = matriz[i][j][0]  // Mantener componente rojo
        rojo[i][j][1] = 0              // Eliminar componente verde
        rojo[i][j][2] = 0              // Eliminar componente azul
      FIN PARA
    FIN PARA
  FIN FUNCION

                FUNCION separar_canal_verde_pnm(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, verde[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    DEFINIR i, j COMO ENTERO
```

```
    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            verde[i][j][0] = 0              // Eliminar componente rojo
            verde[i][j][1] = matriz[i][j][1]   // Mantener componente verde
            verde[i][j][2] = 0              // Eliminar componente azul
        FIN PARA
    FIN PARA
FIN FUNCION

                FUNCION separar_canal_azul_pnm(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,
azul[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    DEFINIR i, j COMO ENTERO

    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            azul[i][j][0] = 0              // Eliminar componente rojo
            azul[i][j][1] = 0              // Eliminar componente verde
            azul[i][j][2] = matriz[i][j][2]   // Mantener componente azul
        FIN PARA
    FIN PARA
FIN FUNCION
```

```
FUNCION convertir_a_grises(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, grises[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO)

    // Declarar variables
    DEFINIR i, j COMO ENTERO
    DEFINIR valor_gris COMO CARACTER_SIN_SIGNO

    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            // Calcular valor de gris usando pesos estándar
            valor_gris = CONVERTIR_A_ENTERO(
                (matriz[i][j][0] * 0.299) +  // Componente rojo (29.9%)
                (matriz[i][j][1] * 0.587) +  // Componente verde (58.7%)
                (matriz[i][j][2] * 0.114)    // Componente azul (11.4%)
            )

            // Asignar el mismo valor a los tres canales
            grises[i][j][0] = valor_gris     // Canal rojo
            grises[i][j][1] = valor_gris     // Canal verde
            grises[i][j][2] = valor_gris     // Canal azul
        FIN PARA
    FIN PARA
FIN FUNCION
```

FUNCION convertir_a_bn(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, bn[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO, umbral: ENTERO)

```
// Declarar variables
DEFINIR i, j COMO ENTERO
DEFINIR valor_gris COMO CARACTER_SIN_SIGNO

// Validar rango del umbral
SI (umbral < 0) ENTONCES
   umbral = 0
FIN SI

SI (umbral > 255) ENTONCES
   umbral = 255
FIN SI

PARA i DESDE 0 HASTA (alto - 1) HACER
   PARA j DESDE 0 HASTA (ancho - 1) HACER
      // Convertir a escala de grises primero
      valor_gris = CONVERTIR_A_ENTERO(
         (matriz[i][j][0] * 0.299) +  // Componente rojo
         (matriz[i][j][1] * 0.587) +  // Componente verde
```

```
            (matriz[i][j][2] * 0.114)    // Componente azul
        )

        // Aplicar umbral para convertir a blanco y negro
        SI (valor_gris >= umbral) ENTONCES
            bn[i][j][0] = 255          // Blanco (canal rojo)
            bn[i][j][1] = 255          // Blanco (canal verde)
            bn[i][j][2] = 255          // Blanco (canal azul)
        SINO
            bn[i][j][0] = 0            // Negro (canal rojo)
            bn[i][j][1] = 0            // Negro (canal verde)
            bn[i][j][2] = 0            // Negro (canal azul)
        FIN SI
      FIN PARA
    FIN PARA
  FIN FUNCION

    FUNCION calcular_histograma_color(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO, histR[256]: ENTERO, histG[256]: ENTERO, histB[256]: ENTERO)

    // Declarar variables
    DEFINIR i, j COMO ENTERO
```

```
    // Inicializar todos los contadores de histogramas a cero
    PARA i DESDE 0 HASTA 255 HACER
        histR[i] = 0    // Inicializar histograma rojo
        histG[i] = 0    // Inicializar histograma verde
        histB[i] = 0    // Inicializar histograma azul
    FIN PARA

    // Contar frecuencia de cada valor de color
    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            histR[matriz[i][j][0]]++    // Incrementar contador para valor rojo
            histG[matriz[i][j][1]]++    // Incrementar contador para valor verde
            histB[matriz[i][j][2]]++    // Incrementar contador para valor azul
        FIN PARA
    FIN PARA
  FIN FUNCION

    FUNCION calcular_histograma_grises(matriz[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO, alto: ENTERO, ancho: ENTERO, hist[256]: ENTERO)

    // Declarar variables
    DEFINIR i, j COMO ENTERO
    DEFINIR valor_gris COMO CARACTER_SIN_SIGNO
```

```
    // Inicializar todos los contadores del histograma a cero
    PARA i DESDE 0 HASTA 255 HACER
        hist[i] = 0
    FIN PARA

    // Calcular y contar valores de gris
    PARA i DESDE 0 HASTA (alto - 1) HACER
        PARA j DESDE 0 HASTA (ancho - 1) HACER
            // Convertir pixel RGB a escala de grises usando pesos estándar
            valor_gris = CONVERTIR_A_ENTERO(
                (matriz[i][j][0] * 0.299) +  // Componente rojo (29.9%)
                (matriz[i][j][1] * 0.587) +  // Componente verde (58.7%)
                (matriz[i][j][2] * 0.114)    // Componente azul (11.4%)
            )

            // Incrementar el contador correspondiente
            hist[valor_gris]++
        FIN PARA
    FIN PARA
FIN FUNCION
```

```
FUNCION escribir_histograma(nombre_archivo: TEXTO CONSTANTE, histograma[256]: ENTERO)

  // Declarar variables
  DEFINIR i, j COMO ENTERO

  // Abrir archivo en modo escritura
  archivo = ABRIR_ARCHIVO(nombre_archivo, "w")

  SI (archivo ES NULO) ENTONCES
     IMPRIMIR "Error al crear archivo ", nombre_archivo
  FIN SI

  // Escribir encabezado con tabuladores
  ESCRIBIR_FORMATO(archivo, "Tono\tValor\tHistograma\n")

  // Escribir datos del histograma
  PARA i DESDE 0 HASTA 255 HACER
     SI (histograma[i] > 0) ENTONCES    // Solo procesar valores que aparecen
        // Escribir tono y valor numérico
        ESCRIBIR_FORMATO(archivo, "%d\t%d\t", i, histograma[i])

        // Escribir representación visual con asteriscos
```

```
        PARA j DESDE 0 HASTA (histograma[i] / 100 - 1) HACER
            ESCRIBIR_FORMATO(archivo, "*")
        FIN PARA

        ESCRIBIR_FORMATO(archivo, "\n")
    FIN SI
  FIN PARA

  CERRAR_ARCHIVO(archivo)
FIN FUNCION

FUNCION mezclar_imagenes(frente[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,fondo[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,mezcla[MAX_ALTO][MAX_ANCHO][3]: CARACTER_SIN_SIGNO,alto: ENTERO,ancho: ENTERO,alpha: ENTERO)

  // Declarar variables
  DEFINIR i, j, k, valor_mezcla COMO ENTERO

  // Validar rango de alpha
  SI (alpha < 0) ENTONCES
    alpha = 0
  FIN SI
```

```
SI (alpha > 255) ENTONCES
   alpha = 255
FIN SI


// Procesar cada pixel
PARA i DESDE 0 HASTA (alto - 1) HACER
   PARA j DESDE 0 HASTA (ancho - 1) HACER
      PARA k DESDE 0 HASTA 2 HACER    // Procesar cada canal RGB
         // Calcular mezcla usando la fórmula:
         // MEZCLAi = (FRENTEi × alpha) ÷ 256 + (FONDOi × (255 − alpha)) ÷ 256
         valor_mezcla = ((frente[i][j][k] * alpha) +
                  (fondo[i][j][k] * (255 - alpha))) / 256

         // Validar rango del resultado
         SI (valor_mezcla > 255) ENTONCES
            valor_mezcla = 255
         FIN SI

         SI (valor_mezcla < 0) ENTONCES
            valor_mezcla = 0
         FIN SI
```

// Asignar valor final

mezcla[i][j][k] = CONVERTIR_A_CARACTER(valor_mezcla)

FIN PARA

FIN PARA

FIN PARA

FIN FUNCION

FIN

Código

```
//Proyecto_final
//Alumnos: Durán Rendón Santiago, Noriega Chiu Santiago, Ramírez Reyes Jesús.
//20-11-24

//Los comentarios de una línea se usaran para la documentación del programa.
/*Los comentarios de multiple línea seran para explicar información que desconocia.*/

/*No se usaran acentos dentro de lo que se imprimira en la terminal ya que luego da
caracteres extraños.*/

//BIBLIOTECAS A USAR
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

//DEFINIR máximo para alto y ancho de la imagen
#define MAX_ANCHO 512//Define un límite máximo para el ancho de la imagen
#define MAX_ALTO 512//Define un límite máximo para el alto de la imagen

//Matrices estáticas para almacenar los canales y declaración de variables.
int alto, ancho, umbral;
int alto_frente, ancho_frente, alto_fondo, ancho_fondo;
unsigned char matriz[MAX_ALTO][MAX_ANCHO][3];
```

```
unsigned char canal_rojo[MAX_ALTO][MAX_ANCHO][3];
unsigned char canal_verde[MAX_ALTO][MAX_ANCHO][3];
unsigned char canal_azul[MAX_ALTO][MAX_ANCHO][3];
unsigned char grises[MAX_ALTO][MAX_ANCHO][3];
unsigned char bn[MAX_ALTO][MAX_ANCHO][3];
int histR[256], histG[256], histB[256], histGris[256];
unsigned char frente[MAX_ALTO][MAX_ANCHO][3];
unsigned char fondo[MAX_ALTO][MAX_ANCHO][3];
unsigned char mezcla[MAX_ALTO][MAX_ANCHO][3];

/*prototipos = Si los prototipos no estuvieran presentes y las funciones estuvieran definidas después del main, el compilador generaría un error o advertencia de que las funciones no están definidas en el momento en que son llamadas.*/
//PROTOTIPOS
void mostrar_ayuda();
void leer_cabecera_bmp(const char *nombreArchivo);
void leer_cabecera_pnm(const char *nombreArchivo);
void leer_imagen_bmp(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int *alto, int *ancho);
void escribir_imagen_bmp(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void separar_canal_rojo_bmp(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char rojo[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void separar_canal_verde_bmp(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char verde[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void separar_canal_azul_bmp(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char azul[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void leer_imagen_pnm(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int *alto, int *ancho);
void escribir_imagen_pnm(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void separar_canal_rojo_pnm(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char rojo[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void separar_canal_verde_pnm(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char verde[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void separar_canal_azul_pnm(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char azul[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void convertir_a_grises(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char grises[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho);
void convertir_a_bn(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char bn[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho, int umbral);
void calcular_histograma_color(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho, int histR[256], int histG[256], int histB[256]);
```

```
void calcular_histograma_grises(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho, int hist[256]);
void escribir_histograma(const char *nombre_archivo, int histograma[256]);
void mezclar_imagenes(unsigned char frente[MAX_ALTO][MAX_ANCHO][3], unsigned char fondo[MAX_ALTO][MAX_ANCHO][3], unsigned char mezcla[MAX_ALTO][MAX_ANCHO][3],int alto, int ancho, int alpha);

//FUNCIÓN PRINCIPAL

/*Cuando ejecutas un programa en C desde la terminal, puedes darle información adicional. Esta información se llama argumentos de línea de comandos, argc y argv son la manera en que el programa recibe esos datos.

-argc: Representa el número de argumentos que se le pasan al programa.

     Ejemplo: Si ejecutas ./programa 1 imagen.bmp, argc tendrá el valor 3 porque hay tres partes en el comando: ./programa, 1, y imagen.bmp.

-argv: Es un arreglo de cadenas de texto (o strings) donde cada posición almacena uno de los argumentos pasados.

    En el mismo ejemplo (./programa 1 imagen.bmp):
        argv[0] sería ./programa (el nombre del programa).
        argv[1] sería 1 (la opción).
        argv[2] sería imagen.bmp (el nombre del archivo de imagen).
/*Cuando ejecutas un programa en C desde la terminal, puedes darle información adicional. Esta información se llama argumentos de línea de comandos, argc y argv son la manera en que el programa recibe esos datos.

-argc: Representa el número de argumentos que se le pasan al programa.

     Ejemplo: Si ejecutas ./programa 1 imagen.bmp, argc tendrá el valor 3 porque hay tres partes en el comando: ./programa, 1, y imagen.bmp.

-argv: Es un arreglo de cadenas de texto (o strings) donde cada posición almacena uno de los argumentos pasados.

    En el mismo ejemplo (./programa 1 imagen.bmp):
        argv[0] sería ./programa (el nombre del programa).
        argv[1] sería 1 (la opción).
        argv[2] sería imagen.bmp (el nombre del archivo de imagen).

Por la explicación anterior es necesaria la biblioteca <stlib.h>*/
```

```c
int main(int argc, char *argv[]){
    //VERIFICAR si se proporcionaron los argumentos necesarios en la línea de comandos.
       //argc representa la cantidad de argumentos ingresados, incluyendo el nombre del programa.
    //argv es un arreglo de cadenas que contiene los argumentos como texto.
    if (argc < 2 || argc > 6){
        printf("Error: No se proporcionaron los argumentos correctos.\n");
        printf("Use el comando -help para obtener mas informacion.\n");
        return 1;/*return 1 significa que no se ejecuto el código como se esperaba.*/
    }//if

    //VERIFICAR si el primer argumento es -help
      /*strcmp = string comparation, los compara por el código ASCCI y devuelve 0 si son iguales, un valor negativo si la primera es menor y un valor positivo si la primera es mayor.*/
    if (strcmp(argv[1], "-help") == 0){/*Comillas dobles "" para comparar char.*/
        mostrar_ayuda();
        return 0;/*Return 0 significa que se ejecuto el código como se esperaba.*/
    }//if

    //LEER cabeceras según el formato
     if (strcmp(argv[1], "1b") == 0 || strcmp(argv[1], "2b") == 0 || strcmp(argv[1], "3b") == 0 || strcmp(argv[1], "4b") == 0 || strcmp(argv[1], "5b") == 0 || strcmp(argv[1], "6b") == 0 || strcmp(argv[1], "7b") == 0 || strcmp(argv[1], "8b") == 0 || strcmp(argv[1], "9b") == 0){
        leer_cabecera_bmp(argv[2]);
      }else if (strcmp(argv[1], "1p") == 0 || strcmp(argv[1], "2p") == 0 || strcmp(argv[1], "3p") == 0 || strcmp(argv[1], "4p") == 0 || strcmp(argv[1], "5p") == 0 || strcmp(argv[1], "6p") == 0 || strcmp(argv[1], "7p") == 0 || strcmp(argv[1], "8p") == 0 || strcmp(argv[1], "9p") == 0){
        leer_cabecera_pnm(argv[2]);
    }//else if

    //VERIFICAR comandos para lectura de separación de canales bmp
    if (strcmp(argv[1], "1b") == 0) {
        //SEPARAR todos los canales
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);

        separar_canal_rojo_bmp(matriz, canal_rojo, alto, ancho);
        separar_canal_verde_bmp(matriz, canal_verde, alto, ancho);
        separar_canal_azul_bmp(matriz, canal_azul, alto, ancho);

        escribir_imagen_bmp("rojo.bmp", canal_rojo, alto, ancho);
        escribir_imagen_bmp("verde.bmp", canal_verde, alto, ancho);
        escribir_imagen_bmp("azul.bmp", canal_azul, alto, ancho);
```

```
        printf("\nSe han creado los archivos rojo.bmp, verde.bmp y azul.bmp.\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "2b") == 0) {
        //SOLO canal rojo
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);
        separar_canal_rojo_bmp(matriz, canal_rojo, alto, ancho);
        escribir_imagen_bmp("rojo.bmp", canal_rojo, alto, ancho);
        printf("\nSe ha creado el archivo rojo.bmp.\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "3b") == 0) {
        //SOLO canal verde
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);
        separar_canal_verde_bmp(matriz, canal_verde, alto, ancho);
        escribir_imagen_bmp("verde.bmp", canal_verde, alto, ancho);
        printf("\nSe ha creado el archivo verde.bmp.\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "4b") == 0) {
        //SOLO canal azul
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);
        separar_canal_azul_bmp(matriz, canal_azul, alto, ancho);
        escribir_imagen_bmp("azul.bmp", canal_azul, alto, ancho);
        printf("\nSe ha creado el archivo azul.bmp.\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "5b") == 0) {
        //CONVERTIR a grises
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);
        convertir_a_grises(matriz, grises, alto, ancho);
        escribir_imagen_bmp("grises.bmp", grises, alto, ancho);
        printf("\nSe ha creado el archivo grises.bmp.\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "6b") == 0) {
        //CONVERTIR a blanco y negro
        //CONVERTIR el umbral de string a entero
            umbral = atoi(argv[3]);/*atoi = convertir una serie de caracteres en un valor
entero.*/
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);
        convertir_a_bn(matriz, bn, alto, ancho, umbral);
        escribir_imagen_bmp("bn.bmp", bn, alto, ancho);
        printf("\nSe ha creado el archivo bn.bmp.\n");
```

```
printf("\n-------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "7b") == 0) {
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);

        //CALCULAR histogramas
        calcular_histograma_color(matriz, alto, ancho, histR, histG, histB);
        calcular_histograma_grises(matriz, alto, ancho, histGris);

        //ESCRIBIR archivos
        escribir_histograma("histR.txt", histR);
        escribir_histograma("histG.txt", histG);
        escribir_histograma("histB.txt", histB);
        escribir_histograma("histGris.txt", histGris);

        printf("Histogramas generados exitosamente\n");

printf("\n-------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "8b") == 0) {
        //VERIFICAR número de argumentos
        if (argc != 5) {
            printf("Error: Uso correcto: %s 8b frente.bmp fondo.bmp alpha\n", argv[0]);
            return 1;
        }//if

        //CONVERTIR alpha a entero
        int alpha = atoi(argv[4]);

        //LEER ambas imágenes
        leer_imagen_bmp(argv[2], frente, &alto_frente, &ancho_frente);
        leer_imagen_bmp(argv[3], fondo, &alto_fondo, &ancho_fondo);

        //VERIFICAR que las imágenes tengan el mismo tamaño
        if (alto_frente != alto_fondo || ancho_frente != ancho_fondo) {
            printf("Error: Las imagenes deben tener el mismo tamano\n");
            return 1;
        }//if

        //REALIZAR la mezcla
        mezclar_imagenes(frente, fondo, mezcla, alto_frente, ancho_frente, alpha);

        //GUARDAR resultado
        escribir_imagen_bmp("mezcla.bmp", mezcla, alto_frente, ancho_frente);
        printf("\nMezcla realizada con exito. Resultado guardado en 'mezcla.bmp'\n");
```

```
printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "9b") == 0) {
        //REALIZAR todos los comandos
        //VERIFICAR argumentos
        if (argc != 6) {
                printf("Error: Uso correcto: %s 9b imagen.bmp fondo.bmp umbral alpha\n",
argv[0]);
            return 1;
        }//if

        int umbral = atoi(argv[4]);
        int alpha = atoi(argv[5]);

        //VERIFICAR rangos
        if (umbral < 0 || umbral > 255 || alpha < 0 || alpha > 255) {
            printf("Error: umbral y alpha deben estar entre 0 y 255\n");
            return 1;
        }//if

        printf("\n=== Procesando todos los comandos para BMP ===\n");

        //1. LEER imágenes y separar canales RGB
        printf("\n1. Separando canales RGB...\n");
        leer_imagen_bmp(argv[2], matriz, &alto, &ancho);
        separar_canal_rojo_bmp(matriz, canal_rojo, alto, ancho);
        separar_canal_verde_bmp(matriz, canal_verde, alto, ancho);
        separar_canal_azul_bmp(matriz, canal_azul, alto, ancho);
        escribir_imagen_bmp("rojo.bmp", canal_rojo, alto, ancho);
        escribir_imagen_bmp("verde.bmp", canal_verde, alto, ancho);
        escribir_imagen_bmp("azul.bmp", canal_azul, alto, ancho);

        //2. ESCALA de grises
        printf("\n2. Generando imagen en escala de grises...\n");
        convertir_a_grises(matriz, grises, alto, ancho);
        escribir_imagen_bmp("grises.bmp", grises, alto, ancho);

        //3. BLANCO y negro
        printf("\n3. Generando imagen en blanco y negro...\n");
        convertir_a_bn(matriz, bn, alto, ancho, umbral);
        escribir_imagen_bmp("bn.bmp", bn, alto, ancho);

        //4. HISTOGRAMAS
        printf("\n4. Calculando histogramas...\n");
```

```
        calcular_histograma_color(matriz, alto, ancho, histR, histG, histB);
        calcular_histograma_grises(matriz, alto, ancho, histGris);
        escribir_histograma("histR.txt", histR);
        escribir_histograma("histG.txt", histG);
        escribir_histograma("histB.txt", histB);
        escribir_histograma("histGris.txt", histGris);

        //5. MEZCLA de imágenes
        printf("\n5. Mezclando imagenes...\n");
        leer_imagen_bmp(argv[3], fondo, &alto_fondo, &ancho_fondo);
        if (alto != alto_fondo || ancho != ancho_fondo) {
            printf("Error: Las imagenes deben tener el mismo tamano\n");
            return 1;
        }//if
        mezclar_imagenes(matriz, fondo, mezcla, alto, ancho, alpha);
        escribir_imagen_bmp("mezcla.bmp", mezcla, alto, ancho);

        printf("\n=== Procesamiento completado ===\n");
        printf("Archivos generados:\n");
        printf("- rojo.bmp, verde.bmp, azul.bmp\n");
        printf("- grises.bmp\n");
        printf("- bn.bmp\n");
        printf("- histR.txt, histG.txt, histB.txt, histGris.txt\n");
        printf("- mezcla.bmp\n");

printf("\n================================================================================\n")
;
    //VERIFICAR comandos para lectura de separación de canales pnm
    } else if (strcmp(argv[1], "1p") == 0) {
        //SEPARAR todos los canales
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);

        separar_canal_rojo_pnm(matriz, canal_rojo, alto, ancho);
        separar_canal_verde_pnm(matriz, canal_verde, alto, ancho);
        separar_canal_azul_pnm(matriz, canal_azul, alto, ancho);

        escribir_imagen_pnm("rojo.pnm", canal_rojo, alto, ancho);
        escribir_imagen_pnm("verde.pnm", canal_verde, alto, ancho);
        escribir_imagen_pnm("azul.pnm", canal_azul, alto, ancho);
        printf("\nSe han creado los archivos rojo.pnm, verde.pnm y azul.pnm.\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "2p") == 0) {
        //SOLO canal rojo
```

```
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
        separar_canal_rojo_pnm(matriz, canal_rojo, alto, ancho);
        escribir_imagen_pnm("rojo.pnm", canal_rojo, alto, ancho);
        printf("\nSe ha creado el archivo rojo.pnm.\n");

printf("\n--------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "3p") == 0) {
        //SOLO canal verde
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
        separar_canal_verde_pnm(matriz, canal_verde, alto, ancho);
        escribir_imagen_pnm("verde.pnm", canal_verde, alto, ancho);
        printf("\nSe ha creado el archivo verde.pnm.\n");

printf("\n--------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "4p") == 0) {
        //SOLO canal azul
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
        separar_canal_azul_pnm(matriz, canal_azul, alto, ancho);
        escribir_imagen_pnm("azul.pnm", canal_azul, alto, ancho);
        printf("\nSe ha creado el archivo azul.pnm.\n");

printf("\n--------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "5p") == 0) {
        //CONVERTIR a grises
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
        convertir_a_grises(matriz, grises, alto, ancho);
        escribir_imagen_pnm("grises.pnm", grises, alto, ancho);
        printf("\nSe ha creado el archivo grises.pnm.\n");

printf("\n--------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "6p") == 0) {
        //CONVERTIR a blanco y negro
        //CONVERTIR el umbral de string a entero
            umbral = atoi(argv[3]);/*atoi = convertir una serie de caracteres en un valor
entero.*/
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
        convertir_a_bn(matriz, bn, alto, ancho, umbral);
        escribir_imagen_pnm("bn.pnm", bn, alto, ancho);
        printf("\nSe ha creado el archivo bn.pnm.\n");

printf("\n--------------------------------------------------------------------------\n");
    }  else if (strcmp(argv[1], "7p") == 0) {
        //SIMILAR al 7b pero usando funciones PNM
        leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
```

```
        //CALCULAR histogramas
        calcular_histograma_color(matriz, alto, ancho, histR, histG, histB);
        calcular_histograma_grises(matriz, alto, ancho, histGris);

        //ESCRIBIR archivos
        escribir_histograma("histR.txt", histR);
        escribir_histograma("histG.txt", histG);
        escribir_histograma("histB.txt", histB);
        escribir_histograma("histGris.txt", histGris);

        printf("Histogramas generados exitosamente\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "8p") == 0) {
        //SIMILAR al 8b pero usando funciones PNM
        if (argc != 5) {
            printf("Error: Uso correcto: %s 8p frente.pnm fondo.pnm alpha\n", argv[0]);
            return 1;
        }//if

        int alpha = atoi(argv[4]);

        leer_imagen_pnm(argv[2], frente, &alto_frente, &ancho_frente);
        leer_imagen_pnm(argv[3], fondo, &alto_fondo, &ancho_fondo);

        if (alto_frente != alto_fondo || ancho_frente != ancho_fondo) {
            printf("Error: Las imagenes deben tener el mismo tamano\n");
            return 1;
        }//if

        mezclar_imagenes(frente, fondo, mezcla, alto_frente, ancho_frente, alpha);
        escribir_imagen_pnm("mezcla.pnm", mezcla, alto_frente, ancho_frente);
        printf("\nMezcla realizada con exito. Resultado guardado en 'mezcla.pnm'\n");

printf("\n--------------------------------------------------------------------------------\n");
    } else if (strcmp(argv[1], "9p") == 0) {
        //REALIZAR todos los comandos
        //SIMILAR al 9b pero usando funciones PNM
        if (argc != 6) {
                printf("Error: Uso correcto: %s 9p imagen.pnm fondo.pnm umbral alpha\n",
argv[0]);
            return 1;
        }//if
```

```
    int umbral = atoi(argv[4]);
    int alpha = atoi(argv[5]);

    // Verificar dimensiones
    if (alto > MAX_ALTO || ancho > MAX_ANCHO ||
        alto_fondo > MAX_ALTO || ancho_fondo > MAX_ANCHO) {
        printf("Error: Las dimensiones exceden los límites permitidos (%dx%d)\n",
        MAX_ALTO, MAX_ANCHO);
        return 1;
    }//if

    if (umbral < 0 || umbral > 255 || alpha < 0 || alpha > 255) {
        printf("Error: umbral y alpha deben estar entre 0 y 255\n");
        return 1;
    }//if

    printf("\n=== Procesando todos los comandos para PNM ===\n");

    //MISMO proceso que 9b pero con funciones PNM
    //1. LEER imágenes y separar canales RGB
    printf("\n1. Separando canales RGB...\n");
    leer_imagen_pnm(argv[2], matriz, &alto, &ancho);
    separar_canal_rojo_pnm(matriz, canal_rojo, alto, ancho);
    separar_canal_verde_pnm(matriz, canal_verde, alto, ancho);
    separar_canal_azul_pnm(matriz, canal_azul, alto, ancho);
    escribir_imagen_pnm("rojo.pnm", canal_rojo, alto, ancho);
    escribir_imagen_pnm("verde.pnm", canal_verde, alto, ancho);
    escribir_imagen_pnm("azul.pnm", canal_azul, alto, ancho);

    //2. ESCALA de grises
    printf("\n2. Generando imagen en escala de grises...\n");
    convertir_a_grises(matriz, grises, alto, ancho);
    escribir_imagen_bmp("grises.pnm", grises, alto, ancho);

    //3. BLANCO y negro
    printf("\n3. Generando imagen en blanco y negro...\n");
    convertir_a_bn(matriz, bn, alto, ancho, umbral);
    escribir_imagen_bmp("bn.pnm", bn, alto, ancho);

    //4. HISTOGRAMAS
    printf("\n4. Calculando histogramas...\n");
    calcular_histograma_color(matriz, alto, ancho, histR, histG, histB);
    calcular_histograma_grises(matriz, alto, ancho, histGris);
```

```
        escribir_histograma("histR.txt", histR);
        escribir_histograma("histG.txt", histG);
        escribir_histograma("histB.txt", histB);
        escribir_histograma("histGris.txt", histGris);

        //5. MEZCLA de imágenes
        printf("\n5. Mezclando imagenes...\n");
        leer_imagen_pnm(argv[3], fondo, &alto_fondo, &ancho_fondo);
        if (alto != alto_fondo || ancho != ancho_fondo) {
            printf("Error: Las imagenes deben tener el mismo tamano\n");
            return 1;
        }//if
        mezclar_imagenes(matriz, fondo, mezcla, alto, ancho, alpha);
        escribir_imagen_pnm("mezcla.pnm", mezcla, alto, ancho);
        //... resto del proceso similar a 9b pero con extensión .pnm ...

        printf("\n=== Procesamiento completado ===\n");
        printf("Archivos generados:\n");
        printf("- rojo.pnm, verde.pnm, azul.pnm\n");
        printf("- grises.pnm\n");
        printf("- bn.pnm\n");
        printf("- histR.txt, histG.txt, histB.txt, histGris.txt\n");
        printf("- mezcla.pnm\n");

printf("\n================================================================================\n")
;
    } else {
        //SI NO se conoce el argumento, mostrar error.
        printf("Error: Comando no reconocido.\n");
        printf("Use el comando -help para obtener mas informacion.\n");
        return 1;
    }//else

    return 0;
}//int main

//FUNCIÓN PARA MOSTRAR LA AYUDA DEL PROGRAMA (-help)
void mostrar_ayuda(){

printf("\n--------------------------------------------------------------------------------
--------------------------\n");//Para separar y mayor comprención al usuario.
    printf("Uso del programa:\n");
    printf("1p o 1b -> Extrar matrices R, G, B de una imagen.\n");
    printf("2p o 2b -> Extraer unicamente la matriz R.\n");
```

```
    printf("3p o 3b -> Extraer unicamente la matriz G.\n");
    printf("4p o 4b -> Extraer unicamente la matriz B.\n");
    printf("5p o 5b -> Generar imagen en escala de grises.\n");
       printf("6p umbral o 6b umbral -> Generar imagen en blanco y negro con umbral
(0-255).\n");
    printf("7p o 7b -> Calcular y guardar el histograma.\n");
    printf("8p alpha o 8b alpha -> Mezclar dos imagenes con coeficiente alpha (0-255).\n");
      printf("9p umbral alpha o 9b umbral alpha -> Realizar todos los procesos sobre las
imagenes.\n");
    printf("-help -> Mostrar este menu de ayuda.\n");

printf("\n----------------------------------------------------------------------------------
--------------------------\n");//Para separar y mayor comprención al usuario.
    printf("\nUso: <tipo imagen> <nombre archivo> [parametros adicionales]\n");
    printf("Ejemplos de uso:\n");
    printf("programa 1p/1p imagen.pnm\n");
    printf("programa 2p/2p imagen.pnm\n");
    printf("programa 3p/3p imagen.pnm\n");
    printf("programa 4p/4p imagen.pnm\n");
    printf("programa 5b/5p imagen.bmp\n");
    printf("programa 6b/6p imagen.bmp umbral\n");
    printf("programa 7b/7p imagen.bmp\n");
    printf("programa 8b/8p frente.pnm fondo.pnm alpha\n");
       printf("programa 9b/9p imagen1.pnm(Esta primera sera la imagen que haga todos los
procesos) imagen2.bmp umbral alpha\n");

printf("\n----------------------------------------------------------------------------------
--------------------------\n");//Para separar y mayor comprención al usuario.
}//void mostrar_ayuda

//LEER CABECERA BMP
void leer_cabecera_bmp(const char *nombre_archivo){/*const =  especifica que el valor de una
variable es constante e indica al compilador que evite que el programador lo modifique.*/
    FILE *archivo = fopen(nombre_archivo, "rb");/*rb = read binary, abrir un archivo binario
existente para lectura.*/
    if (!archivo){/*! = NOT, por tanto si no es el archivo, se ejecuta el error.*/
        printf("Error: No se pudo abrir el archivo %s.\n", nombre_archivo);
        return;/*fuerza una salida inmediata de la función en que se encuentra.*/
    }//if

    //VARIABLES para los campos de la cabecera bmp
    unsigned short tipo, planos, bpp;/*unsigned = sin signo, short =  entero de complemento a
dos con signo de 16 bits . Tiene un valor mínimo de -32.768 y un valor máximo de 32.767, bpp
= bits por pixel.*/
```

```
    unsigned int tamano, reservado, offset, tamanoDIB;//tamano = tamaño.
    int ancho, alto;

    //LEER los campos de la cabecera
      fread(&tipo, sizeof(unsigned short), 1, archivo);/*sizeof = Se utiliza para saber el
tamaño de una variable, tipo de dato, objeto de clases, entre otros.*/
    fread(&tamano, sizeof(unsigned int), 1, archivo);
    fread(&reservado, sizeof(unsigned int), 1, archivo);
    fread(&offset, sizeof(unsigned int), 1, archivo);
    fread(&tamanoDIB, sizeof(unsigned int), 1, archivo);
    fread(&ancho, sizeof(unsigned int), 1, archivo);
    fread(&alto, sizeof(unsigned int), 1, archivo);
    fread(&planos, sizeof(unsigned int), 1, archivo);
    fread(&bpp, sizeof(unsigned int), 1, archivo);

    //MOSTRAR la información de la cabecera bmp

printf("\n-------------------------------------------------------------------------------------
--------------------------\n");//Para separar y mayor comprención al usuario.
    printf("Informacion de la cabecera bmp:\n");
     printf("Tamano del archivo: %u bytes.\n", tamano);/*%u = se usa pra indicar un entero sin
signo.*/
    printf("Dimensiones: %dx%d pixeles.\n", ancho, alto);
    printf("Bits por pixel: %u.\n", bpp);
    printf("Offset de datos: %u bytes.\n", offset);
    printf("\n-----------------------------------------------------------------------\n");

    fclose(archivo);
}//void leer_cabecera_bmp

//LEER CABECERA BMP
void leer_cabecera_pnm(const char *nombre_archivo){
    FILE *archivo = fopen(nombre_archivo, "r");/*r = read*/
    if (!archivo){//! = NOT, por tanto si no es el archivo, se ejecuta el error
        printf("Error: No se pudo abrir el archivo %s.\n", nombre_archivo);
        return;
    }//if

    //VARIABLES para los campos de la cabecera pnm
    char numero_magico[3];
    int ancho, alto, maxval;

    //LEER el número mágico
```

```
      fscanf(archivo, "%2s", numero_magico);/*%s = para indicar una cadena de caracteres, numero mágico = los primeros dos caracteres del archivo PNM.*/

    //SALTAR comentarios, si es que hay claro.
     char c = fgetc(archivo);/*lee un único carácter sin signo de la ruta de entrada en la posición actual y aumenta el puntero de archivo asociado, si lo hay, para que apunte al siguiente carácter.*/
    while (c == '#'){/*Si dentro de la imagen esta el caracter "#", se usan comillas simples para caracteres y cadenas literales.*/
        while (fgetc(archivo) != '\n');//Leer hasta el final de la línea.
        c = fgetc(archivo);
    }//while
    ungetc(c, archivo);//Regresar el caracter que no era un comentario.

    //LEER dimensiones y maxval
    fscanf(archivo, "%d %d %d", &ancho, &alto, &maxval);

    //MOSTRAR la información de la cabecera

printf("\n-----------------------------------------------------------------------------------------------------------\n");//Para separar y mayor comprención al usuario.
    printf("Informacion de la cabecera pnm:\n");
    printf("Numero magico: %s.\n", numero_magico);
    printf("Dimensiones: %dx%d pixeles.\n", ancho, alto);
    printf("Valor maximo: %d.\n", maxval);

printf("\n-----------------------------------------------------------------------------------------------------------\n");//Para separar y mayor comprención al usuario.

    fclose(archivo);
}//leer_cabecera_pnm

//FUNCIONES PARA LEER Y SEPARAR LOS CANALES RGB BMP (esto se vio en una clase de la materia)

/*Todos los ficheros abiertos tienen un indicador para saber por dónde se va en su lectura o escritura. Es el puntero de acceso al fichero. El tipo de este puntero es off_t, pero en realidad en un número entero. Este número indica cuál es la próxima posición del fichero que se va a leer o escribir. La primera posición del fichero tiene el valor 0, la última posición del fichero tiene un valor igual al tamaño del fichero. Para poder saber o modificar la posición actual de un fichero, se usa la llamada al sistema lseek, cuyo prototipo es:
off_t lseek(int fildes, off_t desplazamiento, int dOnde);
Si queremos cambiar la posición del puntero de acceso, realizamos la llamada al sistema con el descriptor del fichero cuyo puntero queremos modificar. Para modificar el puntero se
```

```
indica un desplazamiento (segundo parámetro) tanto positivo como negativo, con respecto a uno
de tres puntos de referencia (tercer parámetro):

Tercer parámetro    Punto de referencia
SEEK_SET            Comienzo del fichero
SEEK_CUR            Punto actual
SEEK_END            Fin del fichero*/

//LEER IMAGEN BMP
void leer_imagen_bmp(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int
*alto, int *ancho) {
    FILE *archivo = fopen(nombre_archivo, "rb");
    //DECLARAR variables
    int bytes_por_linea, padding, i, j;

    //MENSAJE de error
    if (archivo == NULL) {/*si es null no se puede abrir osea se utiliza para indicar que un
puntero no apunta a ningún objeto o función válidos*/
        printf("Error al abrir el archivo %s\n", nombre_archivo);//Mensaje de error.
        return;/*Retornar sin valor ya que es un void.*/
    }//if

    //LEER la cabecera BMP (ignorando valores irrelevantes)
    unsigned char cabecera_bmp[54];
    fread(cabecera_bmp, sizeof(unsigned char), 54, archivo);

    //OBTENER dimensiones de la imagen
    *ancho = *(int*)&cabecera_bmp[18];
    *alto = *(int*)&cabecera_bmp[22];

    //VERIFICAR si las dimensiones exceden los límites establecidos
    if (*ancho > MAX_ANCHO || *alto > MAX_ALTO) {
            printf("Error: La imagen excede los limites permitidos (%dx%d).\n", MAX_ANCHO,
MAX_ALTO);
        fclose(archivo);
        return;
    }//if

    //CALCULAR el padding = bytes por fila (cada línea debe ser múltiplo de 4 bytes)
    bytes_por_linea = (*ancho) * 3;
    padding = (4 - (bytes_por_linea % 4)) % 4;

    //LEER los datos de la imagen
    for (i = *alto - 1; i >= 0; i--) {
```

```
        for (j = 0; j < *ancho; j++) {
            //EN BMP el orden es BGR
            fread(&matriz[i][j][2], 1, 1, archivo);//Azul.
            fread(&matriz[i][j][1], 1, 1, archivo);//Verde.
            fread(&matriz[i][j][0], 1, 1, archivo);//Rojo.
        }//for
        //SALTAR el padding al final de cada línea
        fseek(archivo, padding, SEEK_CUR);
    }//for

    fclose(archivo);
}//void leer_imagen_bmp

//ESCRIBIR IMAGEN BMP
void escribir_imagen_bmp(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3],
int alto, int ancho) {
    FILE *archivo = fopen(nombre_archivo, "wb");
    //DECLARAR variables
    unsigned char cabecera_bmp[54];
    int bytes_por_linea, padding, tamano_archivo, i, j;
    unsigned char byte_padding = 0;

    //MENSAJE de error
    if (archivo == NULL) {
        printf("Error al crear el archivo %s\n", nombre_archivo);
        return;
    }//if

    //CALCULAR padding y tamaño del archivo
    bytes_por_linea = ancho * 3;
    padding = (4 - (bytes_por_linea % 4)) % 4;
    tamano_archivo = 54 + (bytes_por_linea + padding) * alto;

    //PREPARAR cabecera BMP
     memset(cabecera_bmp, 0, 54);/*memset = inicializa los primeros n bytes de la zona de
memoria apuntada por s con el valor de c.*/
    cabecera_bmp[0] = 'B';
    cabecera_bmp[1] = 'M';
    *(int*)&cabecera_bmp[2] = tamano_archivo;
    *(int*)&cabecera_bmp[10] = 54;
    *(int*)&cabecera_bmp[14] = 40;
    *(int*)&cabecera_bmp[18] = ancho;
    *(int*)&cabecera_bmp[22] = alto;
    *(short*)&cabecera_bmp[26] = 1;
```

```
    *(short*)&cabecera_bmp[28] = 24;
    *(int*)&cabecera_bmp[34] = (bytes_por_linea + padding) * alto;

    //ESCRIBIR cabecera
    fwrite(cabecera_bmp, sizeof(unsigned char), 54, archivo);

    //ESCRIBIR datos de la imagen
    for (i = alto - 1; i >= 0; i--) {
        for (j = 0; j < ancho; j++) {
            //ESCRIBIR en orden BGR
            fwrite(&matriz[i][j][2], 1, 1, archivo);//Azul
            fwrite(&matriz[i][j][1], 1, 1, archivo);//Verde
            fwrite(&matriz[i][j][0], 1, 1, archivo);//Rojo
        }//for
        //ESCRIBIR padding
        for (j = 0; j < padding; j++) {
            fwrite(&byte_padding, 1, 1, archivo);
        }//for
    }//for

    fclose(archivo);
}//void escribir_imagen_bmp

//SEPARAR CANAL ROJO
void  separar_canal_rojo_bmp(unsigned  char  matriz[MAX_ALTO][MAX_ANCHO][3],  unsigned  char
rojo[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    int i, j;
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            rojo[i][j][0] = matriz[i][j][0];//Canal rojo se mantiene.
            rojo[i][j][1] = 0;              //Verde a cero.
            rojo[i][j][2] = 0;              //Azul a cero.
        }//for
    }//for
}//void separar_canal_rojo

//SEPARAR CANAL VERDE
void  separar_canal_verde_bmp(unsigned  char  matriz[MAX_ALTO][MAX_ANCHO][3],  unsigned  char
verde[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    int i, j;
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            verde[i][j][0] = 0;              //Rojo a cero.
            verde[i][j][1] = matriz[i][j][1];//Canal verde se mantiene.
```

```
            verde[i][j][2] = 0;              //Azul a cero.
        }//for
    }//for
}//void separar_canal_verde

//SEPARAR CANAL AZUL
void  separar_canal_azul_bmp(unsigned  char  matriz[MAX_ALTO][MAX_ANCHO][3],  unsigned  char
azul[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    int i, j;
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            azul[i][j][0] = 0;              //Rojo a cero
            azul[i][j][1] = 0;              //Verde a cero
            azul[i][j][2] = matriz[i][j][2];//Canal azul se mantiene
        }//for
    }//for
}//void separar_canal_azul_bmp

//FUNCIONES PARA LEER Y SEPARAR LOS CANALES RGB PNM (esto se vio en una clase de la materia)

//LEER IMAGEN PNM
void leer_imagen_pnm(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int
*alto, int *ancho) {
    FILE *archivo = fopen(nombre_archivo, "rb");
    //DECLARAR variables
    char linea[100];
    int max_valor;
    int i, j;

    //MENSAJE de error
    if (archivo == NULL) {
        printf("Error al abrir el archivo %s\n", nombre_archivo);
        return;
    }//if

    //LEER el identificador del formato (P6)
    fgets(linea, sizeof(linea), archivo);
    if (linea[0] != 'P' || linea[1] != '6') {
        printf("Formato de archivo incorrecto\n");
        fclose(archivo);
        return;
    }//if

    //SALTAR comentarios si existen
```

```
    do {
        fgets(linea, sizeof(linea), archivo);
    } while (linea[0] == '#');

    //LEER dimensiones
    sscanf(linea, "%d %d", ancho, alto);

    //LEER valor máximo
    fgets(linea, sizeof(linea), archivo);
    sscanf(linea, "%d", &max_valor);

    //LEER datos de la imagen
    for (i = 0; i < *alto; i++) {
        for (j = 0; j < *ancho; j++) {
            fread(&matriz[i][j][0], 1, 1, archivo);//Rojo
            fread(&matriz[i][j][1], 1, 1, archivo);//Verde
            fread(&matriz[i][j][2], 1, 1, archivo);//Azul
        }//for
    }//for

    fclose(archivo);
}//void leer_imagen_pnm

//ESCRIBIR IMAGEN PNM
void escribir_imagen_pnm(char *nombre_archivo, unsigned char matriz[MAX_ALTO][MAX_ANCHO][3],
int alto, int ancho) {
    FILE *archivo = fopen(nombre_archivo, "wb");
    //DECLARAR variables
    int i, j;

    //MENSAJE de error
    if (archivo == NULL) {
        printf("Error al crear el archivo %s\n", nombre_archivo);
        return;
    }//if

    //ESCRIBIR cabecera
    fprintf(archivo, "P6\n");                //Identificador del formato
    fprintf(archivo, "%d %d\n", ancho, alto);//Dimensiones
    fprintf(archivo, "255\n");               //Valor máximo

    //ESCRIBIR datos de la imagen
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
```

```
            fwrite(&matriz[i][j][0], 1, 1, archivo); // Rojo
            fwrite(&matriz[i][j][1], 1, 1, archivo); // Verde
            fwrite(&matriz[i][j][2], 1, 1, archivo); // Azul
        }//for
    }//for

    fclose(archivo);
}//void escribir_imagen_pnm

//SEPARAR CANAL ROJO
void  separar_canal_rojo_pnm(unsigned  char  matriz[MAX_ALTO][MAX_ANCHO][3],  unsigned  char
rojo[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    int i, j;
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            rojo[i][j][0] = matriz[i][j][0];//Mantener canal rojo
            rojo[i][j][1] = 0;              //Verde a cero
            rojo[i][j][2] = 0;              //Azul a cero
        }//for
    }//for
}//void separar_canal_rojo_pnm

//SEPARAR CANAL VERDE
void  separar_canal_verde_pnm(unsigned  char  matriz[MAX_ALTO][MAX_ANCHO][3],  unsigned  char
verde[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    int i, j;
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            verde[i][j][0] = 0;              //Rojo a cero
            verde[i][j][1] = matriz[i][j][1];//Mantener canal verde
            verde[i][j][2] = 0;              //Azul a cero
        }//for
    }//for
}//void separar_canal_verde_pnm

//SEPARAR CANAL AZUL
void  separar_canal_azul_pnm(unsigned  char  matriz[MAX_ALTO][MAX_ANCHO][3],  unsigned  char
azul[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    int i, j;
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            azul[i][j][0] = 0;              //Rojo a cero
            azul[i][j][1] = 0;              // Verde a cero
            azul[i][j][2] = matriz[i][j][2];//Mantener canal azul
```

```
        }//for
    }//for
}//void separar_canal_azul_pnm

//CONVERTIR A GRIS
void convertir_a_grises(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char
grises[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho) {
    //DECLARAR variables
    int i, j;
    unsigned char valor_gris;

    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            //CALCULAR valor de gris usando los pesos estándar
            valor_gris = (unsigned char)(
                (matriz[i][j][0] * 0.299) + //Rojo
                (matriz[i][j][1] * 0.587) + //Verde
                (matriz[i][j][2] * 0.114)   //Azul
             );//Sumar los valores de los canales, para mayor comprensión se usaron distintos
renglones.

            //ASIGNAR el mismo valor a los tres canales para mantener la imagen en color
            grises[i][j][0] = valor_gris;
            grises[i][j][1] = valor_gris;
            grises[i][j][2] = valor_gris;
        }//for
    }//for
}//void convertir_a_grises

//CONVERTIR A BLANCO Y NEGRO
void convertir_a_bn(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], unsigned char
bn[MAX_ALTO][MAX_ANCHO][3], int alto, int ancho, int umbral) {
    //DECLARAR variables
    int i, j;
    unsigned char valor_gris;

    //VERIFICAR que el umbral esté en el rango válido
    if (umbral < 0) {
        umbral = 0;
    }//if
    if (umbral > 255) {
        umbral = 255;
    }//if
```

```
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            //PRIMERO convertir a escala de grises
            valor_gris = (unsigned char)(
                (matriz[i][j][0] * 0.299) +
                (matriz[i][j][1] * 0.587) +
                (matriz[i][j][2] * 0.114)
            );

            //APLICAR el umbral
            if (valor_gris >= umbral) {
                bn[i][j][0] = 255;//Blanco
                bn[i][j][1] = 255;
                bn[i][j][2] = 255;
            } else {
                bn[i][j][0] = 0;  //Negro
                bn[i][j][1] = 0;
                bn[i][j][2] = 0;
            }//else
        }//for
    }//for
}//void convertir_a_bn

//FUNCIONES PARA CALCULAR Y ESCRIBIR HISTOGRAMAS

//CALCULAR HISTOGRAMA DE IMÁGENES EN COLOR
void calcular_histograma_color(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int alto, int
ancho, int histR[256], int histG[256], int histB[256]) {
    //DECLARAR variables
    int i, j;

    //INICIALIZAR contadores a cero
    for (i = 0; i < 256; i++) {
        histR[i] = 0;
        histG[i] = 0;
        histB[i] = 0;
    }//for

    //CONTAR ocurrencias
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            histR[matriz[i][j][0]]++;
            histG[matriz[i][j][1]]++;
            histB[matriz[i][j][2]]++;
```

```
        }
    }//for
}//void calcular_histograma_color

//CALCULAR HISTOGRAMA DE IMÁGENES EN ESCALA DE GRIS
void calcular_histograma_grises(unsigned char matriz[MAX_ALTO][MAX_ANCHO][3], int alto, int
ancho, int hist[256]) {
    //DECLARAR variables
    int i, j;
    unsigned char valor_gris;

    //INICIALIZAR contadores
    for (i = 0; i < 256; i++) {
        hist[i] = 0;
    }

    //CONTAR ocurrencias
    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            valor_gris = (unsigned char)(
                (matriz[i][j][0] * 0.299) +
                (matriz[i][j][1] * 0.587) +
                (matriz[i][j][2] * 0.114)
            );
            hist[valor_gris]++;
        }//for
    }//for
}//void calcular_histograma_grises

//ESCRIBIR HISTOGRAMA
void escribir_histograma(const char *nombre_archivo, int histograma[256]) {
    FILE *archivo = fopen(nombre_archivo, "w");
    //DECLARAR variables
    int i, j;

    //MENSAJE de error
    if (archivo == NULL) {
        printf("Error al crear archivo %s\n", nombre_archivo);
        return;
    }

    fprintf(archivo, "Tono\tValor\tHistograma\n");/*\t = tabulador*/

    for (i = 0; i < 256; i++) {
```

```
        if (histograma[i] > 0) {//Solo escribir valores que aparecen
            fprintf(archivo, "%d\t%d\t", i, histograma[i]);
            //ESCRIBIR asteriscos
            for (j = 0; j < histograma[i] / 100; j++) { //ESCALAR para mejor visualización
                fprintf(archivo, "*");
            }
            fprintf(archivo, "\n");
        }//if
    }//for

    fclose(archivo);
}//void escribir_histograma

//FUNCIONES PARA MEZCLAR IMÁGENES

void mezclar_imagenes(unsigned char frente[MAX_ALTO][MAX_ANCHO][3], unsigned char
fondo[MAX_ALTO][MAX_ANCHO][3], unsigned char mezcla[MAX_ALTO][MAX_ANCHO][3],int alto, int
ancho, int alpha) {
    //DECLARAR variables
    int i, j, k;
    int valor_mezcla;

    //VERIFICAR que alpha esté en el rango válido
    if (alpha < 0) {
        alpha = 0;
    }//if
    if (alpha > 255) {
        alpha = 255;
    }//if

    for (i = 0; i < alto; i++) {
        for (j = 0; j < ancho; j++) {
            for (k = 0; k < 3; k++) {//Para cada canal RGB
                  //APLICAR la fórmula: MEZCLAi = (FRENTEi × alpha) ÷ 256 + (FONDOi × (255 -
alpha)) ÷ 256
                valor_mezcla = ((frente[i][j][k] * alpha) + (fondo[i][j][k] * (255 - alpha)))
/ 256;

                // Asegurar que el valor esté en el rango 0-255
                if (valor_mezcla > 255) {
                    valor_mezcla = 255;
                }//if
                if (valor_mezcla < 0) {
                    valor_mezcla = 0;
```

```
                }//if

                mezcla[i][j][k] = (unsigned char)valor_mezcla;
            }//for
        }//for
    }//for
}//void mezclar_imagenes
```