Parcial, 2021

Programación para Ingeniería Eléctrica

Jueves 8 de Julio de 2021

La prueba tiene un total de **30** puntos. El mínimo requerido es de **7** puntos (25 % del puntaje total de la prueba). Las preguntas deben responderse de manera lo más concisa posible, respondiéndose sólo lo pedido. A menos que se pida explícitamente, no es necesario ni justificar las respuestas, ni agregar código para que los programas compilen.

Ejercicio 1 - Entrada salida (3 puntos)

Supongamos que tenemos definida una matriz **M** de 3x3 formada por valores flotantes. Es decir es de tipo float[3][3].

Escriba un pedazo de código que imprima en pantalla el valor de la matriz, con 2 cifras decimales de precisión. Los valores de una fila deben aparecer en pantalla separados por 1 espacio y debe saltar de línea al finalizar cada fila.

Ejercicio 2 - Declaraciones y preprocesador (8 puntos)

Escriba un archivo de encabezado `miArchivo.h` con lo siguiente:

- (1 punto) Definir la constante `Epsilon` de valor `0.00001`
- (1 punto) Escriba la **declaración** de una definición de tipo llamado `CodigoError` que sea un enumerado que tome los siguientes valores, en el siguiente orden: TODO_OK, PBL_LECTURA, PBL_ESCRITURA, PBL_MEM y OTRO_PBL.
- (2 puntos) Declare un tipo `Imagen`, que sea una estructura de nombre `imagen` con los siguientes campos (asumimos que será siempre una imagen en grises):
  - `columnas`: de tipo caracter sin signo.
  - `filas`: de tipo caracter sin signo.
  - `pixeles`: de tipo puntero de puntero a caracteres sin signo.
- (3 puntos) Escriba la **declaración** de las siguientes funciones:
  - `FiltrarImagen` que retorne `CodigoError` y tome la siguiente lista de argumentos, en el siguiente orden:
    - Un puntero a un tipo Imagen de nombre `imagenEntrada`
    - Un puntero a un entero, de nombre `tipoFiltro`
    - Un puntero a un tipo Imagen de nombre `imagenSalida`
  - `LeerImagen` que retorne `CodigoError` y que tiene los siguientes parámetros:
    - Un puntero a FILE de nombre `archivoEntrada`

    - Un puntero a un tipo Imagen de nombre `imagenEntrada`
- `LiberarImagen` que retorne `CodigoError` y que tiene el siguiente parámetro:
    - Un puntero a un tipo Imagen de nombre `imagen`

Indique que sentencias del preprocesador c se deben incluir en el `MiArchivo.h` para asegurar su inclusión única en diferentes archivos que lo incluyan. (1 punto)

Ejercicio 3 - Memoria y entrada salida (10 puntos)

Considere que el archivo llamado `archivoEntrada` tiene una imagen en nivel de gris, con la información guardada totalmente en forma binaria y que contiene solo lo siguiente (no hay saltos de línea ni espacio entre cada elemento):

- Dos caracteres sin signo que corresponden al número de filas y el número de columnas respectivamente. Noten que el número de filas y de columnas no puede ser mayor a 256.
- Una sucesión de caracteres sin signo correspondientes a los píxeles de la imagen, que están en niveles de gris.

Escriba la función `LeerImagen` que debe hacer lo siguiente:

- Leer el primer caracter sin signo, que es el número de filas,
- Leer el segundo caracter sin signo, que es el número de columnas,
- Reservar la zona de memoria necesaria para almacenar la imagen que se va a leer,
- Leer desde el `archivoEntrada` los píxeles y almacenarlos en el lugar correspondiente de `imagenEntrada`
- Retornar el valor que corresponda utilizando el enumerado `CodigoError` definido en el ejercicio 2. Recuerden que la función `malloc`, `fread`, etc. retornan cierto valor si hay error. Mirar al respecto información al final.

Recuerden que al finalizar la función, el parámetro `imagenEntrada` debe contener la imagen leída para que desde fuera de la función se pueda acceder a ella.

Ejercicio 4 - Pasaje de parámetros (5 puntos)

1. (3 puntos) Escriba una función de nombre `ProcesaMiImagen`, sin parámetros y sin retorno que:
    - Abra un archivo llamado `MiImagen.pgm` en modo lectura y lo asigne a un puntero a FILE de nombre `pfi`
    - Llame a la función `LeerImagen` definida en el ejercicio 2, pasandole `pfi` como argumento.
    - Cierre el archivo `MiImagen.pgm`.
    - Evalúe el retorno de la función `LeerImagen` y en función del mismo haga alguna de las siguientes acciones:
        - Si es TODO_OK:
            - Llamar a la función `FiltrarImagen`
            - Llamar a la función `LiberarImagen`
        - Si no es TODO_OK
            - Imprimir el mensaje *Hubo un error*,

        - Llamar a la función `LiberarImagen`
    - Salir de la aplicación mediante exit.

2. Realice todas las inclusiones necesarias para que `ProcesarMiImagen` corra suponiendo que está en un archivo diferente a las `LeerImagen`, `FiltrarImagen` y `LiberarImagen`. (1 punto)

3. (1 punto) Explique cómo se podría modificar la declaración de `LeerImagen` y su código para que el nombre del archivo que se lee no sea `MiImagen.pgm` sino una cadena de caracteres que se pase como primer argumento de la función `LeerImagen`

Ejercicio 5 - Argumentos de línea de comandos (4 puntos)

- En este programa la forma del `main` es la usual, de modo que tome una lista de argumentos de la linea de comandos y retorne un entero.
- Escriba un programa `main` que haga lo siguiente:
    - Verifique que el número de argumentos al ejecutar el programa sea igual a 3.
    - de no serlo realice las siguientes operaciones:
        - Imprima el texto `Problema de uso: usted introdujo` *N* `argumentos` donde *N* debe sustituirse por el número de argumentos introducidos (sin incluir el nombre del programa). Se debe incluir el salto de linea al final de lo impreso.
        - Termine el programa con código de salida `-1`.
    - Si el número de argumentos es exactamente 3 entonces:
        - Debe tomar el primer argumento (excluyendo el nombre de programa) como si fuera una cadena de caracteres y abrir en escritura un archivo. Para ello declare y utilice un puntero a FILE de nombre pfi.
        - Debe tomar el segundo argumento (excluyendo el nombre de programa) como si fuera un entero y asignarlo a la variable entera llamada *valor*.
        - Cerrar el puntero a archivo pfi.
        - Termine el programa con código de salida `0`.

Funciones de la biblioteca estándar

A continuación la firma de algunas funciones de la biblioteca estándar que pueden ser de utilidad en los ejercicios. De algunas además está la descripción detallada de lo que hacen más abajo.

```c
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);

FILE *fopen(const char *path, const char *mode);
int fclose( FILE *stream);

int fgetc(FILE *stream);
int fputc(int c, FILE *stream);

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

void *malloc(size_t size);
void free(void *ptr);

void exit(int status);
```

`fopen()` Abre el fichero cuyo nombre es la cadena apuntada por `path` y asocia un flujo de datos a él.

El argumento `mode` apunta a una cadena que empieza con una de las siguientes secuencias (a las que pueden seguir caracteres adicionales):

r Abre un fichero de texto para lectura. El flujo se posiciona al principio del fichero.

r+ Abre para lectura y escritura. El flujo se posiciona al principio del fichero.

w Trunca el fichero a longitud cero o crea un fichero de texto para escritura. El flujo se posiciona al principio del fichero.

w+ Abre para lectura y escritura. El fichero se crea si no existe, en otro caso se trunca. El flujo se posiciona al principio del fichero.

a Abre para añadir (escribir al final del fichero). El fichero se crea si no existe. El flujo se posiciona al final del fichero.

a+ Abre para leer y añadir (escribir al final del fichero). El fichero se crea si no existe. El flujo se posiciona al final del fichero.

Esta función retorna un puntero a FILE si salió todo bien. De lo contrario, retorna `NULL`.

`fclose()` Cierra el flujo de archivo apuntado por `stream`.

`fgetc()` Lee el siguiente carácter de `stream` y lo devuelve como un `unsigned char` convertido a `int`, o `EOF` al llegar al final del flujo o en caso de error.

`fputc()` Escribe el carácter `c`, convertido a `unsigned char`, en el archivo apuntado por `stream`.

`malloc()` Asigna `size` bytes y devuelve un puntero a la memoria asignada. La memoria no es inicializada. Si `size` es 0, `malloc()` retorna `NULL` o bien un valor de puntero único que puede ser pasado a `free()` sin problemas. Si el pedido de memoria falla, la función retorna `NULL`.

`free()` Libera el espacio de memoria apuntado por `ptr`, que debe haber sido devuelto por una llamada previa a `malloc()`, `calloc()` o `realloc()`. En caso contrario, o si `free(ptr)` ya se ha llamado antes, se produce un comportamiento indefinido. Si `ptr` es `NULL`, no se realiza ninguna operación.

VALOR DEVUELTO: Estas funciones devuelven el número de elementos de la entrada asignados, que pueden ser menores que los formatos suministrados para conversión, o incluso cero, en el caso de un fallo de concordancia. Cero indica que, mientras había caracteres disponibles en la entrada, no ocurrió ninguna asignación; normalmente esto es debido a un carácter de entrada inválido,

como un carácter alfabético para una conversión '%d'. Se devuelve el valor EOF si ha habido un fallo de entrada antes de ninguna conversión, como cuando se llega al final de la entrada. Si ocurre un error de lectura o se llega al final de la entrada después de que se haya hecho alguna conversión al menos, se devuelve el número de conversiones completadas hasta ese punto con éxito.

`fread()` lee `nmemb` items de data, cada uno de tamaño `size` bytes, desde el puntero al flujo apuntado por `stream` y lo guarda en el lugar apuntado por `ptr`. Devuelve el número de items leídos y si hubo un error devuelve el valor zero.