Parcial, 2022

Programación para Ingeniería Eléctrica

8 de Julio de 2022

La prueba tiene un total de **36** puntos. El mínimo requerido es de **9** 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 las cosas compilen.

Duración del parcial: 3hs.

Ejercicio 1 - Operaciones de bits (4.5 puntos)

Considere la representación de píxeles RGB utilizada en el obligatorio. Escriba una función:
`int escala_grises(unsigned int pixel);`
Que reciba un `unsigned int pixel` con el formato de la Figura 1, y retorne un `int` con el valor de ese pixel en escala de grises según la fórmula:

$$gris = 0,2 \times r + 0,7 \times g + 0,1 \times b$$

Donde r, g y b son los valores de brillo de cada canal del pixel en la representación RGB.

no usado	rojo	verde	azul
31 . . . 24	23 . . . 16	15 . . . 8	7 . . . 0

Figura 1: Representación de un píxel RGB en un entero.

En caso de necesidad debe castear el valor de retorno.

Ejercicio 2 - Operaciones de bits (4.5 puntos)

Escribir la siguiente función:
`unsigned int concatena(unsigned int buffer_ms, unsigned int buffer_ls, int num_bits_ls);`
No se puede utilizar loops de iteración (`for`, `while`, `do-while`, etc) ni arreglos. Deben utilizar exclusivamente operaciones de bits.

Esta debe concatenar los `num_bits_ls` bits menos significativos de `buffer_ls`, en `buffer_ms`. Esto es, se debe crear un `unsigned int` *resultado* que contenga los `num_bits_ls` bits menos significativos de `buffer_ls` en sus `num_bits_ls` bits menos significativos, y a partir del bit `num_bits_ls` contenga los bits de `buffer_ms`, comenzando por el menos significativo. Se debe suponer que $1 \leq$ `num_bits_ls` ≤ 31, en caso contrario el comportamiento es indeterminado.

Ejemplo: si `buffer_ms`=0x00099999, `buffer_ls`=0x00006666 y `num_bits_ls`=10, `resultado`=0x26666666 como se muestra en la Figura 2.

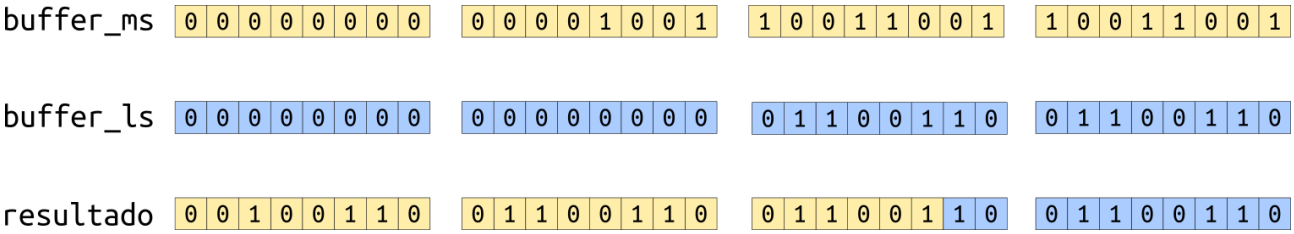

Figura 2: Ejemplo con num_bits_ls = 10.

Ejercicio 3 - Estructuras (4.5 puntos)

Crear un tipo `Imagen_t`, en base a una estructura de nombre `imagen` con los siguientes campos:

- `filas`: de tipo `int`
- `columnas`: de tipo `int`
- `pixeles`: de tipo puntero doble a `unsigned int`

Ejercicio 4 - Punteros y reserva de memoria (4.5 puntos)

Escribir una función que inicialice los datos de una imagen como la vista en el obligatorio apuntada por `pin` y reserve memoria para sus nuevos píxeles (según la Figura 3). No se debe inicializar los pixeles con ningún valor en particular. En caso de falla en los pedidos de memoria con `malloc` retornar de la función.

```
void inicializar_imagen(Imagen_t* pin, int filas, int columnas);
```

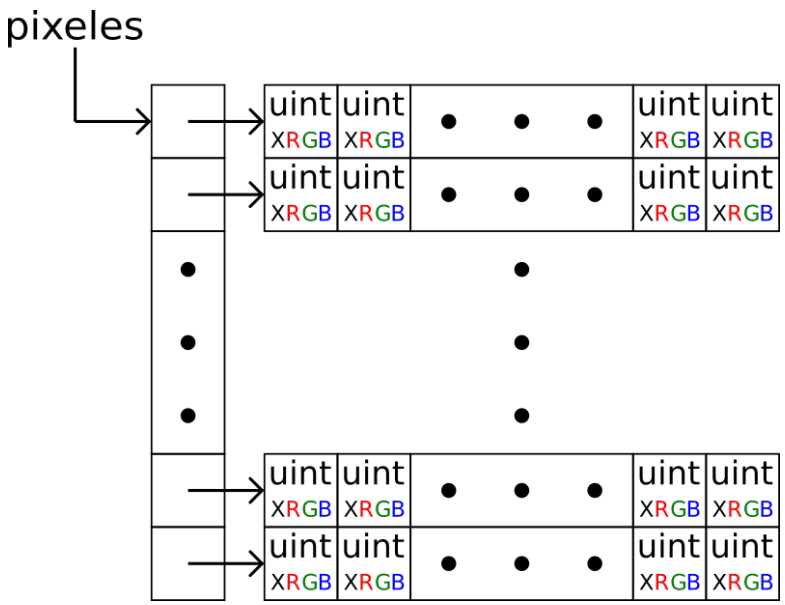

Figura 3: Representación en memoria de `pin->pixeles`.

Ejercicio 5 - Lectura de datos (4.5 puntos)

Escriba la siguiente función:

```
int es_cuadrada(const char * ruta_archivo);
```

Dada la ruta de un archivo en que **únicamente** contiene el encabezado de un archivo PPM Plano (ver Anexo), debe leer el encabezado de la imagen y verificar si es cuadrada, retornando 1 en caso de serlo y 0 en caso contrario.

La función debe hacer lo siguiente:

1. Abrir el archivo `ruta_archivo` en modo lectura; si hay error al abrir, retornar -1.
2. Leer el numero mágico, en caso de que sea incorrecto cerrar el archivo y retornar -1.
3. Leer la cantidad de filas y columnas de la imagen. Se asume que estas siguen el formato correcto y no es necesario contemplar un error.
4. Cerrar el archivo.
5. Verificar si es cuadrada y retornar 1 en caso de que la imagen sea cuadrada, y 0 en caso contrario.

Notar que no se pide hacer nada con el valor máximo de canal, por lo cual no es necesario leerlo ni hacerle verificaciones adicionales.

Ejercicio 6 - Argumentos de línea de comandos (4.5 puntos)

a) Declare el encabezado usual de la función `main` de modo que tome una lista de argumentos de la linea de comandos y devuelva un `int` como resultado.

b) Escriba un fragmento de código que verifique que el número de argumentos **excluyendo el nombre de programa** (*N*) sea **par** y en tal caso imprima:

`‘‘usted introdujo N argumentos’’`

No debe escribir el `main` completo; escriba sólo el fragmento de código que haga lo pedido, asumiendo que está dentro del cuerpo del `main` declarado en la primera parte.

Ejercicio 7 - Escritura binaria (4.5 puntos)

Escriba la siguiente función:

```
void escribir(const unsigned char* datos, const int n, const char* ruta_archivo);
```

La función debe:

1. Abrir el archivo de nombre `ruta_archivo` en modo escritura; si hay error al abrir el archivo, retornar de la función.
2. Escribir cada uno de los `n` valores del puntero `datos` en el archivo en **forma binaria** (no ASCII).
3. Cerrar el archivo.

Ejercicio 8 - Encriptación (4.5 puntos)

a) Explique brevemente qué se entiende por un algoritmo de encriptación y cual es el objetivo. En particular defina entrada, encriptación, criptograma y des-encriptación.

b) Describa resumidamente qué se entiende por un algoritmo de encriptación de clave simétrica.

Funciones de la biblioteca estándar

```c
int printf(const char *format, ...);
FILE *fopen(const char *path, const char *mode);
int fclose( FILE *stream);
int fprintf (FILE* stream, const char * format, ...);
int fscanf(FILE *stream, const char *format, ...);
int fgetc(FILE *stream);
int fputc(int c, FILE *stream);
int strcmp (const char* str1, const char* str2);
void *malloc(size_t size);
```

`printf()` Escribe una string en stdout (salida estándar). La string proporcionada puede contener especificadores de formato (comenzando con `%`). Si hay especificadores de formato, estos se reemplazan con sus respectivos argumentos que siguen a string a la llamada printf. Estos especificadores de formato también pueden contener su longitud, precisión y otras banderas. Algunos especificadores de formato típicos son `%d` para entero, `%c` para caracter y `%f` para punto flotante.

`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`.

`fscanf()` Lee texto formateado de un puntero a archivo. En `stream` se pasa el puntero a archivo del cual se leerá el flujo, mientras que en `format` se pasa un string formateado con eventuales especificadores y banderas como en `printf`.

`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`.

`strcmp()` Compara dos strings carácter por carácter. Si los strings son iguales, la función devuelve 0.

`malloc()` Asigna `size` bytes y devuelve un puntero a la memoria asignada. La memoria no es inicializada. Si `size` es 0, `malloc()` retorna `NULL`. Si el pedido de memoria falla, la función retorna `NULL`.

Encabezado PPM Plano

1. Un *número mágico* para identificar el tipo de archivo. El número mágico de una imagen *PPM plano* es el par de caracteres ASCII `P3`.
2. Uno o más espacios en blanco (espacios, tabuladores, caracter de nueva linea).
3. El ancho (cantidad de columnas) n de la imagen, descrito como una cadena de caracteres ASCII, por ejemplo 123.
4. Uno o más espacios en blanco.
5. El alto (cantidad de filas) m de la imagen, de nuevo en ASCII.
6. Uno o más espacios en blanco.
7. El valor máximo de canal (M), de nuevo en decimal ASCII. Nosotros asumiremos que las imágenes a procesar tendrán como valor máximo a 255.