# Primer parcial

Parte teórica

1. Todo programa puede escribirse utilizando únicamente 3 estructuras básicas para el control de la programación. Mencione cada una ellas y dibuje su diagrama de flujo.   
     
   Las estructuras básicas son: secuencial, selectiva, repetitiva.

/\*Diagrama de flujo adjunto en hoja entregada.\*/

1. ¿En qué consiste el diseño **TOP-DOWN** y cuáles son sus ventajas?  
     
   Consiste en la descomposición de un problema en subproblemas más simples. Sus ventajas son que mejoran mucho la legibilidad, se pueden reutilizar las funciones. Además las funciones se pueden escribir, compilar y depurar en forma independiente…En un programa de gran tamaño pueden trabajar distintos programadores. Se puede modificar (actualizar) una función sin afectar el resto del programa
2. A) Explique qué significa pasar parámetros por valor y por referencia. B) ¿Los arreglos pasan por referencia o por valor como parámetro de una función? C) Si al pasar un arreglo como parámetro es necesario asegurar que **NO** se modifiquen sus elementos, ¿Existe alguna forma en Lenguaje C de indicárselo al compilador?
3. Cuando se pasa una variable por valor se pasa una copia de la variable original a la función y lo que se modifique en ella, se modifica en esa copia, la variable original no se modifica. Cuando se pasa por referencia, se pasa la dirección en memoria en la que está guardada esa variable, al trabajar con la dirección en memoria, si modificamos el valor que toma la variable y lo guardamos en la misma dirección de memoria sobrescribimos la variable original.
4. Por referencia.
5. Sí, utilizando el modificador “const”. Tanto en la declaración como en la implementación de la función a través de la cual estamos pasando un arreglo, cuando se pone el tipo de dato del arreglo, se le agrega un const adelante.

Ejemplo: void imprimir\_arreglo (const arr []).

1. Explique el significado del modificador de ámbito ***static***. De ejemplos de su uso en lenguaje C cuando se utiliza en la declaración de una variable y una función.

Mantiene el valor de una variable dentro de su ámbito (bloque, función o archivo). Cuando se aplica a una variable local su valor se retiene entre sucesivas llamadas a la función (se inicializa automáticamente la primera vez con 0 por defecto). Como modificador de funciones hace que la función sólo se puede acceder desde el archivo donde se declaró.

1. Modifique el siguiente fragmento de código C para que la función *printf()* muestre el resultado 165:

int8\_t a = 115;

int8\_t b = 50;

uint8\_t r = 0; /\*Cambie el tipo de int8\_t a uint8\_t\*/

r = a + b;

printf("%d", r); // 165

1. A) Justifique la salida del siguiente fragmento de código C para la entrada especificada:

int32\_t i, j;

float k;

scanf("\n %d %f %d", &i, &j, &k); // ENTRADA: 5.2 7 8

printf("\n %d %d %e", i, j, k);

// Imprime 5 1045220557 9.809089e-045

Por el formato del scanf, primero busca un entero, después un flotante y después un entero nuevamente. Entonces:

i = 5;

j = 0.2;

k = 7;

\_ Como i es de tipo entero, coincide con el formato del printf, por lo cual imprime como i = 5.

\_ Con j, primero utiliza el standard IEEE-754 para convertir el 0.2 a float. Se guarda en memoria como 0XCDCC4C3E (En memoria se guarda al revés por byte, 🡪 0x3E4CCCCD). Una vez hecho eso, como j no es de tipo float, ese número (3E4CCCCD que es el 0.2 pasado a hexadecimal por el formato IEE-754) lo pasa a decimal, y este se guarda en la variable j. Por lo cual imprime j = 1045220557.

\_Con k, nuevamente utiliza el standard IEEE-754 para convertir 7 a float. Se guarda en memoria como: 0x07000000 (En memoria se guarda al revés por byte. 🡪 0x00000007). Una vez hecho eso, ese número (00000007 que es el 7 pasado a hexadecimal por el formato IEE-754) se guarda en k. Por lo cual, k = 9.80908925027372e-45.

Una vez asignados todos los valores:

Imprime ‘\n’ y un espacio.

Imprime i en formato decimal: 5

Imprime espacio.

Imprime j en formato decimal: 1045220557

Imprime espacio.

Imprime k en formato exponencial: 9.809089e-045 (no imprime todos los decimales porque es de tipo float y no tiene tanta precisión).

B) Indique que queda en el buffer y porqué.

“ 8”. Quedan un espacio y un 8 porque el scanf cortó en 7 (por todo lo explicado en el inciso A).

1. A) Explique por qué la salida de los printf tienen esa forma (siendo el último tramo, valores basura que existen en memoria) B) Introduzca todas las correcciones necesarias para que el código C muestre solo las salidas:

hola

unodostres

uint8\_t cadena1[4] = {0x68,0157,0x6C,0x61};

uint8\_t cadena2[10] = "unodostres";

printf("\n %s", cadena1);

printf("\n %s", cadena2);

// Imprime:

// holaqQkòÜº▄~Ä'v║æ¢û£

// unodostresholaqQkòÜº▄~Ä'v║æ¢û£

1. Porque en la primera cadena no tiene el ‘\0’ y en la segunda cadena, no tiene la cantidad de elementos necesarios declarados, ya que falta un índice para el ‘\0’.
2. Una de las correciones sería simplemente agrandar ambas dimensiones y agregar el ‘\0’:

uint8\_t cadena1[5] = {0x68,0157,0x6C,0x61, 0x00};

uint8\_t cadena2[11] = "unodostres"; //Acá no se agrega, el carácter demás automáticamente es un ‘\0’.

1. A) Ejecute y estudie la salida del siguiente fragmento de código B) Explique cuál es el objetivo de ejecutar una operación XOR con la expresión (1 << n):

uint8\_t x = 0x0A; // 0000 1010

uint8\_t y = 0xFF; // 1111 1111

y = x ^ (1 << 1); // 0000 1010

//^ 0000 0010

//-----------

// 0000 1000 === 0x08

y = y ^ (1 << 3); // 0000 1000

//^ 0000 1000

//-----------

// 0000 0000 === 0x00

printf("\n 1) 0x%02x", y);

y = y ^ (1 << 1); // 0000 0000

//^ 0000 0010

//-----------

// 0000 0010 === 0x02

y = y ^ (1 << 3); // 0000 0010

//^ 0000 1000

//-----------

// 0000 1010 === 0x0A

printf("\n 2) 0x%02X", y);

Se usa el XOR para cambiar el valor y luego volverlo a la normalidad?

1. A continuación se muestra un algoritmo implementado en lenguaje C para mostrar un arreglo 1D de 12 elementos como si fueran arreglos 2D de 3x4 y 4x3 filas y columnas. Implemente una función que reciba como parámetro el arreglo “m” y otros parámetros, de forma que al invocarla dos veces se obtenga el mismo resultado. Además, dicha función deberá servir para mostrar el arreglo “m” como matriz de 6x2 y 2x6.

uint8\_t m[12] = {1,2,3,4,5,6,7,8,9,1,2,3};

printf("\n");

for (int i = 0; i < 12;i++)

{

if (i % 4 == 0)

printf("\t");

printf("%d ", m[i]);

if ((i+1) % 4 == 0)

printf("\n");

}

printf("\n");

for (int i = 0; i < 12;i++)

{

if (i % 3 == 0)

printf("\t");

printf("%d ", m[i]);

if ((i+1) % 3 == 0)

printf("\n");

}

1. Explique qué hace el siguiente algoritmo implementado en lenguaje C:

uint8\_t b[2] = {255, 0377};

uint8\_t uc = 0x1A;

uint8\_t m = 017;

b[1] = uc & m;

b[0] = (uc >> 4) & m;

/\*Asigna al elemento 1 del vector un nuevo valor.

B[1] = 11010

&

01111

--------

01010 = 10 = A

B[0] = (uc >> 4) & m

Uc >> 4 = (1A) >> 4 = 11010 >> 4 = 00001

00001

&

01111

-----------

00001 = 1 = 0x01.

\*/

for (int i=0; i<2 ;i++)

{

printf("\n 0x%02x", b[i]); //La salida del printf es que ocupe 2 espacios nada mas y rellenados con 0 si no los ocupa el valor que va en forma hexadecimal y antes el 0x, ej A = 0x0a (ademas la x esta en minuscula, entonces imprime la A en minuscula).

}

Parte práctica

Se requiere implementar un algoritmo en lenguaje C que permita transformar una imagen color en otra imagen blanco y negro. Un desarrollador que comenzó a realizar este trabajo no llegó a terminarlo pero dejó un par de diagramas de flujo ( Figura 1: Flujo principal y Figura 2: Matriz descompuesta), los cuales se muestran a continuación. Se sabe que la imagen color se debe almacenar en memoria en una estructura secuencial de 1D, cuyos elementos son todos positivos y ocupan 4 bytes de memoria. El primer byte de cada elemento es el nivel de opacidad de la imagen completa y su valor será 0x00 siempre. Los 3 bytes restantes de cada elemento representan las componentes RGB del color de ese elemento. El algoritmo que usted debe implementar deberá hacer lo siguiente:

1. Descomponer cada elemento de la imagen color en sus 4 componentes y almacenar la imagen descompuesta en una nueva estructura de datos.
2. Mostrar al usuario la estructura obtenida en el punto anterior, como se muestra en la Figura 3: Salida del programa.
3. Transformar la imagen obtenida en el punto 1 en una imagen blanco y negro, siguiendo el algoritmo: nivel\_gris = rojo \* 0.30 + verde \* 0.59 + azul \* 0.11. El valor obtenido deberá ser redondeado y tendrá que estar contenido en el intervalo [0 255].
4. Mostrar al usuario la imagen blanco y negro resultante, como se muestra en la Figura 3: Salida del programa.

### Diagramas

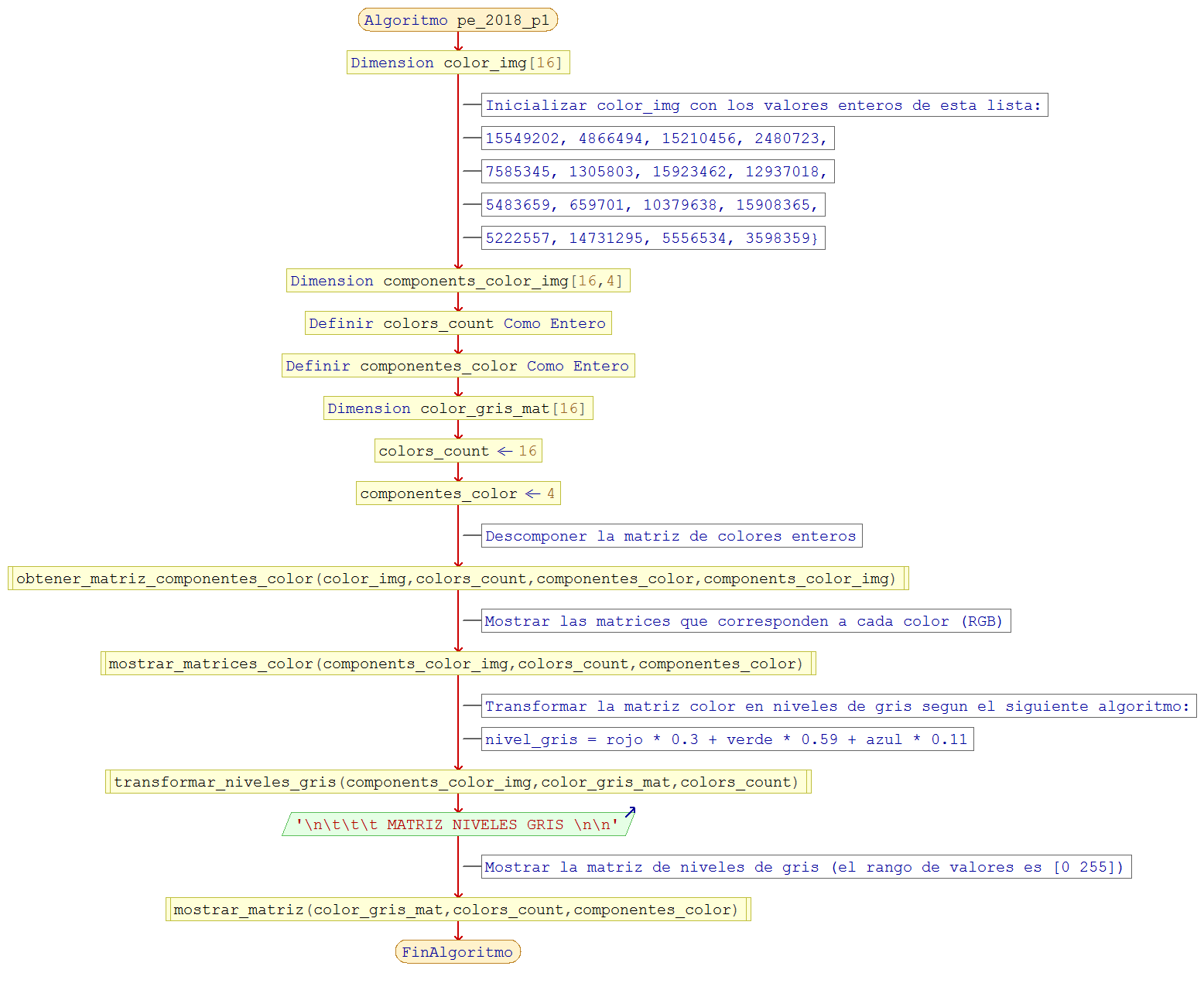


Figura 1: Flujo principal

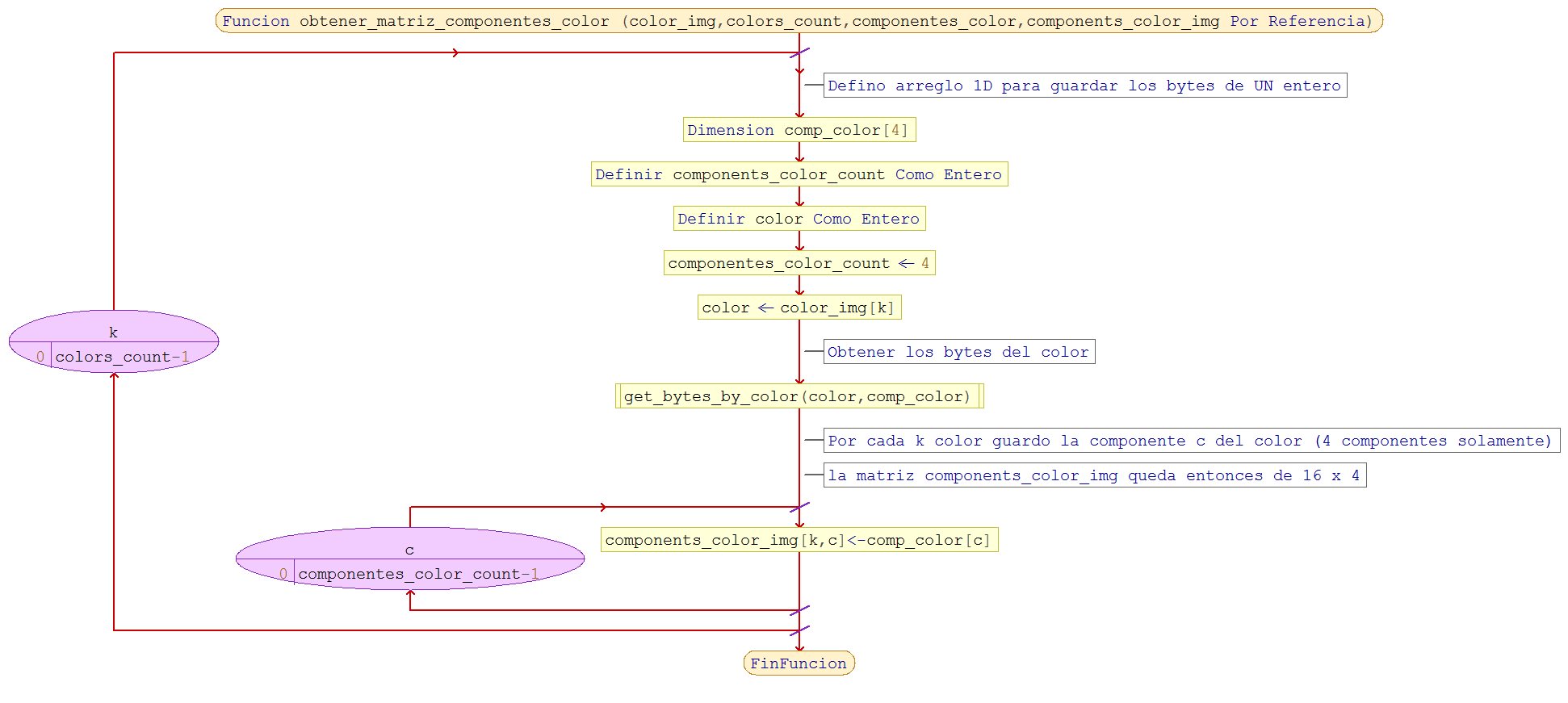


Figura 2: Matriz descompuesta

### Salida

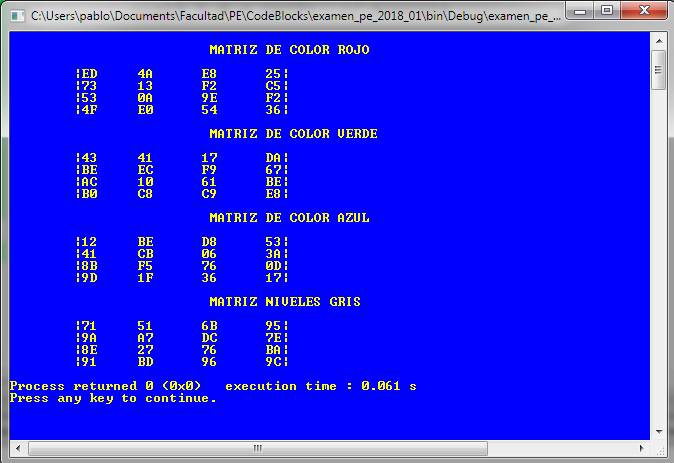


Figura 3: Salida del programa

## Aclaraciones

* Lea atentamente los enunciados de cada ejercicio. Durante la primera h se contestarán las dudas que surjan de los enunciados. Recuerde que primero deberá saber QUE hacer y luego pensar COMO hacerlo.
* Deberá entregar junto al proyecto de CodeBlocks este mismo documento, con las respuestas a las preguntas teóricas y el diagrama de flujo que corresponde a la función que permite obtener los bytes de un tipo de dato uint32\_t.
* Si rinde el examen en una PC de escritorio recuerde guardar el proyecto de CodeBlocks en la unidad D: y que el path que no contenga espacios ni caracteres especiales como la ñ o las vocales acentuadas. Esto último es igual para las notebooks.
* El programa entregado puede estar en un único archivo, no es necesario en esta instancia separar las declaraciones e implementaciones de las funciones en archivos diferentes. Por supuesto que el programa deberá compilar sin errores ni advertencias. La única advertencia tolerada es la que surge de pasar un arreglo 2D declarado NO constante a una función que recibe un parámetro const de ese mismo tipo.
* Los valores de la matriz color\_img[16] son: 15549202, 4866494, 15210456, 2480723, 7585345, 1305803, 15923462, 12937018, 5483659, 659701, 10379638, 15908365, 5222557, 14731295, 5556534, 3598359

**Exitos! =)**