Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas

Organización del Computador 2
Primer parcial – 07/05/2019

1 (40)	2 (40)	3 (20)	

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

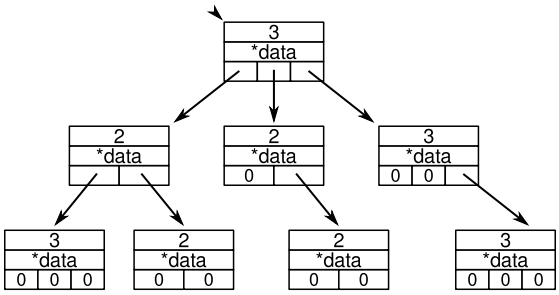
Ej. 1. (40 puntos)

Considerar un árbol construido utilizando para los nodos las siguientes dos estructuras:

```
typedef struct s_nodoDos {
    char tipo;
    void* data;
    void* derecha;
    void* izquierda;
} nodoDos;
```

```
typedef struct s_nodoTres {
    char tipo;
    void* data;
    void* derecha;
    void* izquierda;
    void* centro;
} nodoTres;
```

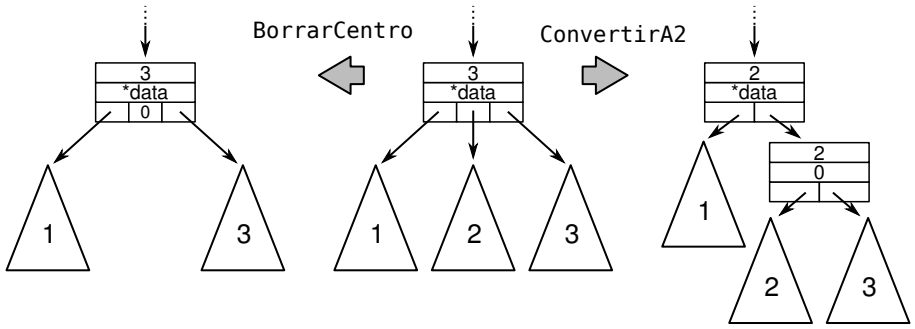
Ejemplo:



El árbol puede tener nodos de cualquiera de los dos tipos, para identificar si el nodo almacena dos o tres punteros, se utiliza el primer byte de la estructura que indica con un `2` o `3` de que tipo es respectivamente.

(20p) a. Implementar la función `void BorrarCentro(void **nodo, funBorrar *fb)`, que toma un doble puntero a un nodo, recorre el árbol y por cada nodo de tipo 3, borra todos los nodos del subarbol apuntado desde el puntero `centro`. Además, utiliza la función `fb` para borrar todos los datos que son eliminados.

(20p) b. Implementar la función `void ConvertirA2(void **nodo)`, que toma un doble puntero a un nodo, recorre el árbol y por cada nodo de tipo 3 realiza la siguiente acción: Convierte al nodo de tipo 3 a tipo 2 conservando el dato original y el subarbol izquierdo. Además, como hijo derecho agrega un nuevo de tipo 2 cuyos subarboles izquierdo y derecho serán los subárboles central y derecho, respectivamente, del nodo original reemplazado. El dato del nuevo nodo debe ser un puntero a *null*.

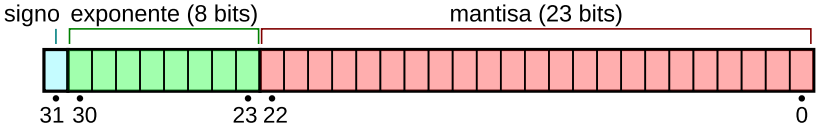


Ej. 2. (40 puntos)

(20p) a. Implementar la función `double* sumasRestas(float *A, short n)` que toma un arreglo de `n` números en punto flotante de simple precisión, realiza la operación a continuación y almacena el resultado en el mismo arreglo pero como punto flotante de doble precisión.

$$dst[i] = \frac{src[2 \cdot i] + src[2 \cdot i + 1]}{src[2 \cdot i + 1] - src[2 \cdot i]} \qquad \text{con } i \text{ entre 0 y } (n-1)/2$$

(20p) b. Implementar la función `void remplazarExponentes(float *A, short n, uint8_t *exp)` que dado un arreglo de `n` números en punto flotante de simple precisión y un arreglo de `n` bytes, remplaza los exponentes de cada uno de los valores de punto flotante por los almacenados en el arreglo de bytes.



Nota: considerar que los arreglos de ambos items tiene tamaño múltiplo de 16.

Ej. 3. (20 puntos)

Un nuevo entorno de compilación y linkeo permite llevar registro de la cantidad de veces que una función es llamada. Los contadores de cada función son almacenados como un número de 64bits exactamente antes del comienzo del código de la función.

Para incrementar estos contadores, se reemplazó la instrucción `ret` en las funciones por `jmp newRet`. Esta última es la encargada de incrementar el contador de la función y retornar de la misma a la función desde donde fue llamada.

(15p) a. Implementar la función `newRet`. Considerar que todas las funciones son llamadas por medio de la instrucción `call IMM`. La codificación de la misma es:

···	1	2	...	9	···
···	opcode	dirección de la función			···

(5p) b. Si una función pudiera ser llamada utilizando diferentes codificaciones de la instrucción call. ¿Es posible implementar la función `newRet`?