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

Organización del Computador 2
Recuperatorio del Primer Parcial – 02/07/2019

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

Normas generales

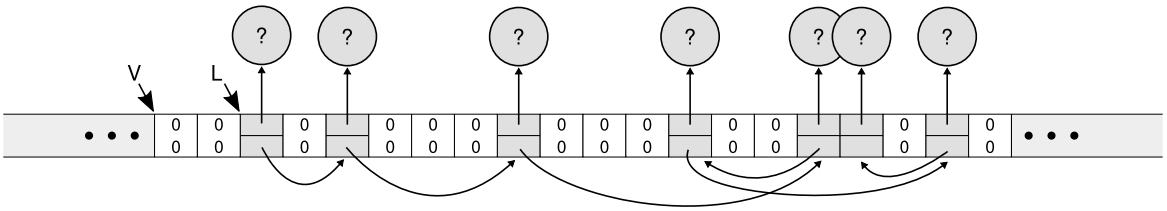
- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. 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)

Se tiene un vector de nodos de tamaño n, que respeta la siguiente estructura.

```
typedef struct s_node_t {
    void *data;
    struct s_node_t *next;
} node_t;
```

Estos nodos a su vez son utilizados para construir una lista simplemente enlazada. Los nodos de esta lista siempre pertenecen a nodos en el rango del vector. Tanto el vector como la lista se identifican por un puntero al inicio y al primer nodo respectivamente.



(20p) a. Implementar la función `int esParaAdelante(nodo* v, nodo* l, int n)` que recorre la lista de nodos y determina si el siguiente a cada nodo de la lista tiene mayor *offset* en el vector. Es decir, el *offset* del nodo en el vector es siempre una dirección de memoria mayor a la del nodo anterior. Retorna un 0 si la lista cumple la condición y 1 en caso contrario.

(20p) b. Implementar la función `int agregarNodoIesimo(nodo* v, nodo** l, int n, int i, void* data)` que, de ser posible, agrega un nodo a la lista a partir de la posición `i` de la lista. Debe buscar un nodo libre en el vector, agregar el dato pasado por parámetro al nodo, y luego agregar el nodo a la lista. Para el caso `i=0` se debe además modificar el doble puntero para que contenga un puntero al nuevo primer nodo de la lista. En caso de poder agregar el nodo solicitado, la función debe retornar un 0, caso contrario un 1.

Nota: Considerar que los nodos que no forman parte de la lista tienen sus dos punteros en *null*.

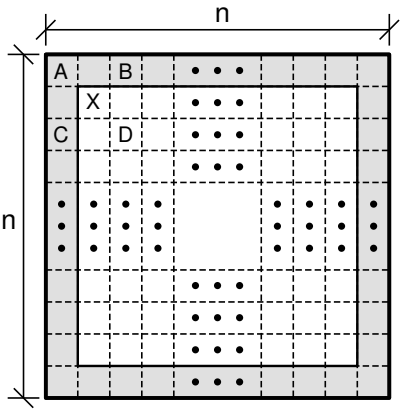
Ej. 2. (40 puntos)

Considerar una imagen en escala de grises de pixeles de 1 byte, de tamaño $n \times n$.

Se pide escribir una función que aplique el operador descripto a continuación de dos formas:

operador

1	0	1
0	0	0
1	0	1



(A) $$O_{i,j} = (I_{(i+1,j+1)} + I_{(i+1,j-1)} + I_{(i-1,j+1)} + I_{(i-1,j-1)})/4$$

(B) $$O_{i,j} = \sqrt{I_{(i+1,j+1)}{}^2 + I_{(i+1,j-1)}{}^2 + I_{(i-1,j+1)}{}^2 + I_{(i-1,j-1)}{}^2}$$

donde I es la imagen fuente y O la imagen destino.

La aridad de la función pedida es: `uint8_t* f(uint8_t* I, uint16_t n)`. Esta debe solicitar memoria para la nueva imagen que contendrá **solamente** los elementos procesados, es decir, no contendrá el borde.

(5p) a. Explicar detalladamente la estrategia que utilizaría para procesar la imagen y evitar casos borde, considerando que se desea procesar al menos 4 pixeles simultáneamente por iteración. Indicar qué propiedad debe cumplir n para ello.

(20p) b. Implementar la función utilizando el operador (A).

(15p) c. Indicar cámo modificar la función anterior para utilizar el operador (B).

Nota: La implementación se debe realizar en SIMD calculando 4 elementos simultáneamente en ambos casos. Ademaś, la instrucción `SQRTPS` calcula la raíz cuadrada.

Ej. 3. (20 puntos)

Un curioso entorno de programación respeta una "casi convención C". La diferencia está en que **NO** alinea la pila previo al llamado de una función. A raíz de esto, si se requiere llamar a una función que deba utilizar la pila alineada, entonces se debe ejecutar código que la alinee:

```
f:
; Codigo inicial
    call alinearPila
; Codigo que usa la pila alineada
    call restaurarPila
; Codigo final
    ret
```

Se pide implementar dos funciones, una denominada `alinearPila` y su contraparte `restaurarPila`. La primera se encarga de detectar si el puntero al tope de la pila debe ser modificado o no, mientras que la segunda restaura la acción realizada por la primera sobre la pila.

(20p) a. Implementar las funciones pedidas considerando que la alineación de la pila solicitada es a 32 bytes. Realizar además un gráfico del estado de la pila antes y despues de llamar a las funciones.

Nota: Una dirección esta alineada a 16 bytes si en su representación binaria, sus 4 bits menos significativos son cero.