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

Organización del Computador 2
Recuperatorio del primer parcial – 05/07/2011

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)

Escriba una función en lenguaje ensamblador que dada una matriz `A` de elementos de 8 bits con signo de dimensión $n \times n$ (n de 16 bits sin signo) aplique el filtro `F`, guardando el resultado en la matriz `B` (de elementos de 16 bits con signo).

El prototipo de la función es: `void aplicarFiltro(char *A, short *B, unsigned short n)`

`F=`

2	-sqrt(2)	2
-sqrt(3)	4	-sqrt(3)
2	-sqrt(2)	2

Aclaraciones:

- La función se debe implementar utilizando instrucciones SIMD y se deben procesar por lo menos 4 elementos simultáneamente.
- No se debe perder precisión en los cálculos (salvo por las conversiones de punto flotante a entero).
- Fuera del rango en el cual se puede aplicar el filtro, no se debe modificar la matriz B.
- Puede asumir que n es múltiplo de un valor conveniente para no tener que manejar casos bordes (indicar el múltiplo elegido).
- En cada instrucción SSE se debe mostrar el contenido del registro destino.

Ej. 2. 40 puntos

La intersección de dos árboles, `A` y `B`, es un árbol `C` que está formado por los nodos que `A` y `B` tienen en común. Dos nodos son comunes entre ambos árboles si están al mismo nivel y tienen la misma clave.

Se desea realizar una función que interseque dos árboles, es decir, que devuelva el árbol intersección de los árboles que recibe como parámetros. Los nodos de los árboles sobre los que opera esta función pueden tener varios hijos. Más precisamente, cada nodo del árbol cuenta con una lista de hijos.

Las estructuras de datos utilizadas son las siguientes:

```
typedef struct _nodo_arbol {
      int clave;
      float dato;
      nodo_lista *hijos;
} __attribute__ ((__packed__)) nodo_arbol;

typedef struct _nodo_lista {
```

```
      nodo_arbol *arbol;
      struct _nodo_lista *proximo;
} __attribute__ ((__packed__)) nodo_lista;
```

El prototipo de la función es: `nodo_arbol *intersecar(nodo_arbol *A, nodo_arbol *B)`

1. (10 ptos) Escribir el pseudocódigo de la función.
2. (30 ptos) Escibir la función en lenguaje ensamblador.

Aclaraciones:

- Las claves de los nodos son únicas en cada árbol.
- El campo `dato` de cada nodo del árbol intersección debe ser igual al campo `dato` del nodo correspondiente en el árbol `A`.
- Cuando los árboles tienen un nodo en común, los hijos de ese nodo en el árbol resultante son los nodos en común de los hijos de los nodos en común de los árboles originales. Por ejemplo, si los árboles originales no tienen la raíz en común, el arbol resultante es vacío.
- Se puede asumir que cuando el puntero `hijos` no es nulo, contiene al menos un puntero `arbol` válido.
- Tenga en cuenta que ciertas secciones del código que escriba pueden ser reutilizadas.

Ej. 3. (20 puntos)

Se pide construir una función con la siguiente aridad,

`treeCall(void* f, nodo_arbol* A, unsigned int k)`

donde `k` es un entero, `f(...)` es una función de `k` parámetros y `A` es un árbol (con la misma estructura que el ejercicio anterior).

La función `treeCall()` se encarga de retornar el resultado de ejecutar la función `f(...)` con los parámetros tomados del árbol `A`.

La forma de obtener los parámetros consiste en tomar los datos de las `k` primeras hojas del árbol. Por suerte, la función `dameLasHojasDelArbol(nodo_arbol* A, float* buffer, unsigned int k)` se encarga de este trabajo, almacenando en un vector los datos de las hojas del árbol.

1. (15 ptos) Implemente en lenguaje ensablador la función `treeCall`.
2. (5 ptos) Dibuje el estado de la pila desde antes del llamado a `treeCall`, hasta la dirección de retorno del llamado a `dameLasHojasDelArbol`.

Aclaración: El orden en que la función `dameLasHojasDelArbol` almacena los datos de las hojas es el mismo en que deben ser almacenadas en la pila para llamar a la función `f(...)`.