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

Organización del Computador 2
Primer parcial – 12/05/2015

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)

Los templos griegos del periodo clásico suelen ser de planta rectangular rodeados de columnas. Estos respetan la siguiente regla sobre la cantidad de columnas de sus lados: 2N+1=M, siendo M el lado largo y N el lado corto.

Se tiene una lista simplemente encadenada que almacena en cada nodo las características de un templo, nombre y cantidad de columnas de cada lado.

```
typedef struct {
  unsigned char colum_largo;
  char *nombre;
  templo *siguiente;
  unsigned char colum_corto;
} templo;
```

Se pide construir una función que tome una lista y reordene los nodos de la misma en dos listas, de forma que una de estas contenga los nodos que respeten la regla y otra los nodos que no la respeten.

a- (6p) Indicar los desplazamientos dentro de la estructura, notar que no es `__packed__`.

b- (6p) Plantear la aridad de la función a realizar. Justificar el porqué de los parámetros.

c- (28p) Escribir la función pedida.

d- (10p) Explique cómo modificar la función anterior si se deben borrar los nodos que no respetan la regla.

Ej. 2. (40 puntos)

El analisis estático de fuerzas es una disciplina que permite estudiar las cargas que afectan una construcción. El objetivo final es determinar si una construcción se encuentra en equilibrio. Como las fuerzas involucradas suelen ser muchas y tener distintas magnitudes se optó por un formato para almacenarlas donde cada componente tiene distinto tamaño.

```
struct fuerza {
  int8_t x;
  int16_t y;
  int24_t z;
} __attribute__((packed));
```

Se posee un vector de 102 fuerzas que afectan a un sistema. Se busca determinar si el sistema se encuentra en equilibrio. Para calcular si un sistema esta en equilibrio se debe calcular su resultante mediante la siguiente formula:

$$R = \sqrt{\left(\sum_{i=1}^{102} x_i\right)^2 + \left(\sum_{i=1}^{102} y_i\right)^2 + \left(\sum_{i=1}^{102} z_i\right)^2}$$

Si la resultante es cero, entonces el sistema se encuentra en equilibrio.

a- (26p) Construir una función que tome un puntero al vector de fuerzas, retorne 1 si el sistema se encuentra en equilibrio y 0 en caso contrario.

b- (8p) ¿Cómo modificaría el código anterior si todas las componentes fueran de tipo `int16_t`?

c- (6p) ¿Cómo modificaría el código anterior si la cantidad de fuerzas a componer fuera impar?

Nota: Todos las componentes son enteros **con signo**. Considerar el tipo `int24_t` como un entero con signo de 24bits. Resolver el ejercicio utilizando instrucciones de SIMD y operando con las tres componentes al mismo tiempo.

Ej. 3. (20 puntos)

Un obstinado desarollador de software está utilizando una compleja biblioteca de funciones desarrollada en un extraño y desconocido lenguaje de programación. Esta biblioteca tiene errores, por lo que el astuto desarrollador planteó un experimento para analizar todo el árbol de llamadas a funciones.

El experimento consta de reemplazar en el código compilado todas las instrucciones `call` x, por la instrucción `call log_call`. La función `log_call` tiene como objetivo loggear datos del llamado antes de hacerlo propiamente. Para ello se encargará de realizar los tres pasos siguientes:

1. Obtener la dirección de la función original (es decir la dirección de x)
2. Almacenar en el log de llamadas datos para el seguimiento
3. Llamar a la función x, como si nunca se hubiera llamado a `log_call`

Para los pasos 1 y 2 se proveen las siguiente funciones que deberán utilizar:

1. `void* get_func(void* addr_call)`
   Retorna el puntero a la función original dada la posición de memoria de la instrucción `call`.
2. `void store_data(void* rsp, void* rbp, void* func)`
   Almacena en el log de llamadas los valores que tenían `rsp` y `rbp` justo antes de la llamada a la función x, y el puntero `func` obtenido mediante la función anterior.

a- (20p) Programar en ASM la función `log_call`. Recordar que se debe respetar el estado del programa original en todo momento.

Nota: Considerar que la instrucción `call` ocupa exactamente 10 bytes. Además las funciónes dadas respetan convención C, con la salvedad que no afectan los registos `xmm1` a `xmm15`.