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

# Organización del Computador 2
# Recuperatorio del Primer Parcial – 4/12/2012

| 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.
- Existen tres notas posibles para los parciales: 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. Para los recuperatorios existen sólo dos notas posibles: I: 0 a 64 pts y A: 65 a 100 pts.

## Ej. 1. (40 puntos)

Marvina, una estresada estudiante de computación, decide regularmente ir a pasar sus tardes al segundo piso. Para llegar, utiliza el ascensor que, al no estar en óptimas condiciones, tarda cierto tiempo $t$ en reaccionar después de que se presiona un botón.

Incapáz de relajarse, cada vez que se toma al ascensor, Marvina percibe este tiempo y lo anota, obteniendo así una lista de tiempos. Los tiempos los anota en unidades enteras de décimas de segundo.

Marvina es fanática de las listas_marv. En una lista_marv se cumple que entre dos elementos consecutivos la diferencia es siempre la misma.

Para obtener una lista_marv a partir de una lista cualquiera, primero hay que eliminar los elementos que no pertenecen (se considera que los dos primeros elementos, de existir, siempre pertenecen), y luego agregar elementos para completar. Se desea que la lista_marv tenga la misma longitud que la lista de la cual se parte.

Marvina desea saber cúal es la mínima cantidad de elementos que deberían insertarse en la lista para obtener una lista_marv usando el procedimiento anterior. Además desea obtener esta lista_marv resultante.

Marvina pensó algunos ejemplos y la siguiente estructura para su lista

| Lista original | lista_marv resultante | Resultado |
|---|---|---|
| 1 5 8 4 | 1 5 9 13 | 2 |
| 1 1 2 3 5 2 6 | 1 1 1 1 1 1 1 | 5 |
| -8 -7 -6 -15 0 | -8 -7 -6 -5 -4 | 2 |

```
typedef struct nodo_t
{
    int elemento;
    struct nodo_t *siguiente;
} __attribute__((__packed__)) nodo;
```

```
typedef struct lista_t
{
    nodo *primero;
    int longitud;
} lista;
```

La función a implementar es `int lista_marv(lista* l);` donde el valor de retorno representa la cantidad de elementos agregados. La lista l se modifica, pero no cambia la cantidad de elementos que tiene.

1. (10p) Escribir el pseudo-código de la función `lista_marv`.
2. (30p) Implementar en ASM de 64 bits la función `lista_marv`.

# Ej. 2. (40 puntos)

Se cuenta con dos matrices de números de 8 bits sin signo, *fuente* y *destino*, ambas de tamaño *alto* × *ancho*. Cada elemento de la matríz *destino* se forma a partir de la matriz *fuente* de la siguiente forma:

$$\text{destino}(i,j) = \begin{cases} \text{0xFF} & \text{si menores}(i,j) \geq 1 \\ \text{0x00} & \text{en otro caso} \end{cases}$$

donde:

$$\text{menores}(i,j) = \text{cantidad de vecinos}^1 \text{ de fuente}(i,j) \text{ menores o iguales a } \texttt{umbral}^2.$$

La aridad de la función a implementar es:

```
void filtrar_menores(unsigned char* destino, unsigned char *fuente,
   int alto, int ancho, unsigned char umbral)
```

(10p) 1. Indicar para qué valores de $i, j$ es posible aplicar este filtro, y qué propiedad deben tener los valores *alto* y *ancho* para que pueda procesarse la mayor cantidad de píxeles posibles por iteración utilizando SIMD.

(30p) 2. Implementar en ASM de 64 bits, utilizando SIMD, el filtro pedido; considerando las restricciones del item anterior. Indicar el contenido de cada registro `xmm` en cada instrucción.

[1] Para cada $(i,j)$, se consideran como vecinos: fuente$(i+1,j)$ y fuente$(i-1,j)$.

[2] `umbral` es un valor pasado por parámetro que no se modifica.

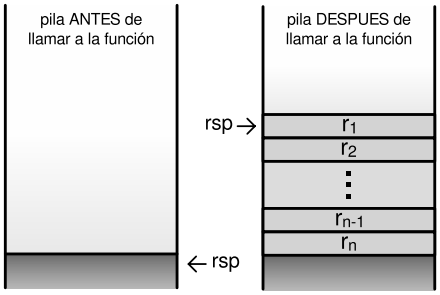
# Ej. 3. (20 puntos)

Una extraña y muy poco usada extensión de `C`, modifica la forma en que se realizan los llamados a funciones. El nuevo mecanismo provee a las funciones de la capacidad de escribir y modificar la pila de la función llamadora.

Las aridades de las funciones cambian agregando una lista de valores de retorno.

`((` $r1$ `, ... ,`$r_n$ `)) nombre (` $p1$ `, ... ,`$p_n$ `)`

Las funciones deben poder escribir en la pila los valores de retorno listados en su aridad.
Realizan esta acción de la siguiente forma: ⟶




1. Implementar, respetando la convención anterior, una función que tome dos numeros enteros de 64 bits, los incremente en 1 y los retorne en la pila.
   La aridad de esta función es: `((int64, int64)) incda1(int64, int64)`.
2. Considerando la siguiente estructura recursiva:

```
struct nodo{
     float dato1;
     float dato2;
     struct nodo *derecho;
     struct nodo *izquierdo;
} __attribute__((__packed__));
```

construir una función que la recorra recursivamente, y retorne la sumatoria de los dos datos (1 y 2) por separado. La aridad de esta función es: `((float sum_dato1, float sum_dato2)) incda2(nodo *primero)`.