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

Organización del Computador 2
Primer parcial – 14/05/2013

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)

Dado un árbol con la estructura descripta a continuación, se quieren mover algunos de sus nodos. Para determinar si un nodo se moverá, se evalúa el elemento de dicho nodo con la función `seConvierteA-Hoja()` (ya dada). Esta evaluación se hará una única vez por nodo, cuando el estado de dicho nodo sea `noEvaluado`.

Si la función devuelve `convertir`, entonces se deben tomar las acciones necesarias para la transformación. Éstas son:

1. destruir el nodo evaluado,
2. crear uno **nuevo** (respetando los datos originales),
3. ubicarlo al final de su lista de hermanos,
4. covertir a todos sus hijos en hermanos, ubicándolos en la posicion donde estaba el nodo originalmente.

El orden de arbolización debe ser depth-first pre-order. Además, luego de que un nodo se convierte a hoja, debe continuarse la arbolización por sus ex-hijos (que a partir de ese momento son sus hermanos). Recuerde los casos borde: si el nodo convertido no tenía hijos, deberan unirse su antecesor con el primer hijo, el hermano siguiente o él mismo, según corresponda. El antecesor será el hermano anterior o, si no tuviera, su padre. Si el nodo convertido no tiene hermanos, igual deberá crearse un nuevo nodo.

La estructura es:

```
typedef enum accion {noConvertir = 0, convertir = 1} accion;
typedef enum estado {noEvaluado  = 0, evaluado  = 1} estado;

typedef struct nodo {
    short  elem;
    estado estado;
    nodo   *hermano;
    nodo   *hijo;
} __attribute__((__packed__)) nodo;
```

(40p) La función a implementar es:

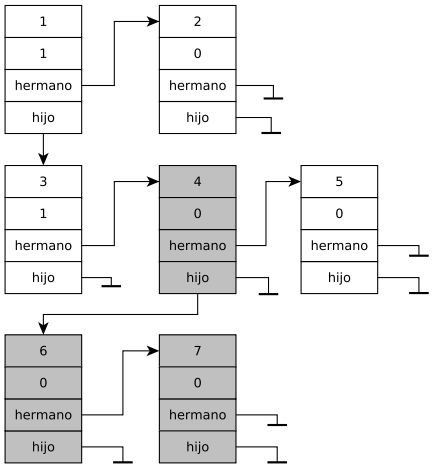
```
nodo* arbolizar(nodo* arbol, accion (*seConvierteAHoja)(short *elem));
```

Ésta función deberá ser capaz de realizar lo pedido para todos los nodos de un árbol; es decir, cada nodo, sus hermanos y sus hijos.

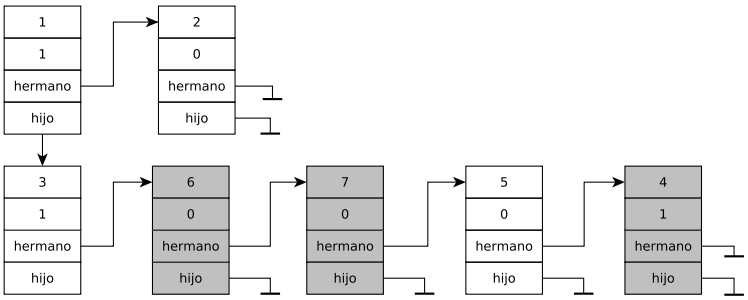
La función pasada por parámetro tiene la aridad `accion seConvierteAHoja(short* elem)` y retorna un valor de tipo enumerado, representando la acción a seguir. La función `arbolizar` debe retornar la raíz del árbol, ya que ésta puede cambiar.

(10p) (a) Escribir el pseudo-código de la función `arbolizar`.

(30p) (b) Implementar en ASM de 64 bits la función `arbolizar`.



(a) Arbol previo a arbolizar el nodo de color con valor 4.



(b) Arbol luego de arbolizar el nodo de color con valor 4.

Ej. 2. (40 puntos)

Dado un vector $x = (x_1, \cdots, x_n)$, definimos el promedio como

$$\mu = \sum_{i=1}^{n} \frac{x_i}{n}$$

Además, definimos la desviación estándar como:

$$\sigma = \sqrt{\sum_{i=1}^{n} (x_i - \mu)^2}$$

Se cuenta con una matriz de números de 8 bits con signo de tamaño *alto* $\times$ *ancho* ($n \times m$). Se desea implementar el cálculo de la desviación estándar de cada columna de la matriz con índice par. La aridad de la función a implementar es:

```
void desviacion_estandar(char *matriz, unsigned int n, unsigned int m, float *promedios, float *std);
```

donde `promedios` es un arreglo con los promedios ya calculados de las filas pares, y `std` es el arreglo de salida donde se escriben las desviaciones estándar de las columnas pares.

(5p) (a) Escribir la función desviacion_estandar en C.

(25p) (b) Escribir la función desviacion_estandar en assembler, optimizando mediante el uso de instrucciones SSE. Asumir que la matriz tiene filas y columnas (n y m) múltiplo de 128, con n y m >32.

(10p) (c) Modificar el código del inciso (b) para el caso de que la matriz tenga n y m cualesquiera, pero mayores a 32.

Ej. 3. (20 puntos)

Definimos la funcion de fibonacci como:

$$fib(0) = 0 \quad fib(1) = 1 \quad fib(n+2) = fib(n) + fib(n+1)$$

Los programadores de orga II encontraron bajo un bloque de hielo en la antartida la siguiente implementación de fibonacci:

```
int fibonacci_forward(int n)
{
    int v;
    if (cantidad() == 0) {
        v = 0;
    } else if (cantidad() == 1) {
        v = 1;
    } else {
        v = ultima() + anteultima();
    }

    if (cantidad() == n)
        return v;
    else
        return fibonacci_forward(n);
}
```

sin embargo, no han podido encontrar las implementaciones de las funciones `cantidad`, `ultima` y `anteultima`.

(5p) (a) Implementar en assembler las funciones `ultima` y `anteultima`, que deben obtener los valores de la última y anteúltima iteración realizada respectivamente.

(10p) (b) Implementar en assembler la funcion `cantidad`, que determina cuantas iteraciones se han realizado hasta el momento (se cuenta con las etiquetas `fibonacci_forward` y `fin_fibonacci_forward`, que se refieren a las direcciones inicial y final de `fibonacci_forward`).

(5p) (c) ¿Qué suposiciones hace sobre el código de `fibonacci_forward`? Justifique detalladamente.