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

Organización del Computador 2
Recuperatorio del Primer Parcial – 12/07/2012

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.
- No se permite el uso de dispositivos electrónicos que no pertenezcan al siguiente conjunto: { calculadora }.
- 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 posee un programa en Fortran que genera un buffer con datos climáticos, este almacena los datos como enteros de 16bits en *big endian*. Lamentablemente para alimentar nuestro modelo climático es necesario convertir estos datos a float en *little endian* y sumarles un coeficiente.

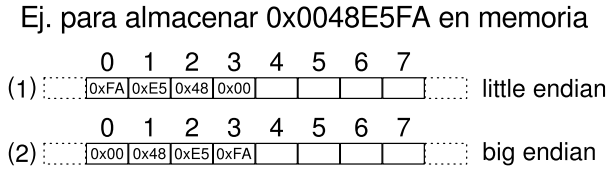
Se desea programar en ASM la función:

`float* convertData(unsigned int count, short* datos, float* coefs)` ,

que toma la cantidad de elementos de los dos vectores (`count`), y los punteros a los vectores de `datos` y `coeficientes`. La función debe retornar un nuevo vector con los valores de los vectores sumados elemento a elemento.

1. (30 puntos) Construir la función pedida considerando que `count` es un número múltiplo de 8.
2. (10 puntos) Proponga una modificación a su código teniendo en cuenta que `count` no es múltiplo de 8 y que los datos son enteros sin signo de 4 bytes.

Nota: En *big endian* el byte más significativo del número ocupa la posición más pequeña de memoria y el menos significativo, la posición más grande de memoria.



Ej. 2. (40 puntos)

Dada una estructura de tabla de Hash se busca realizar una función que elimine un elemento de la misma. Para saber la ubicación de un elemento en la tabla de hash, se provee una función que permite obtener el hash, con la siguiente aridad:

`short int evalHash(void*)`

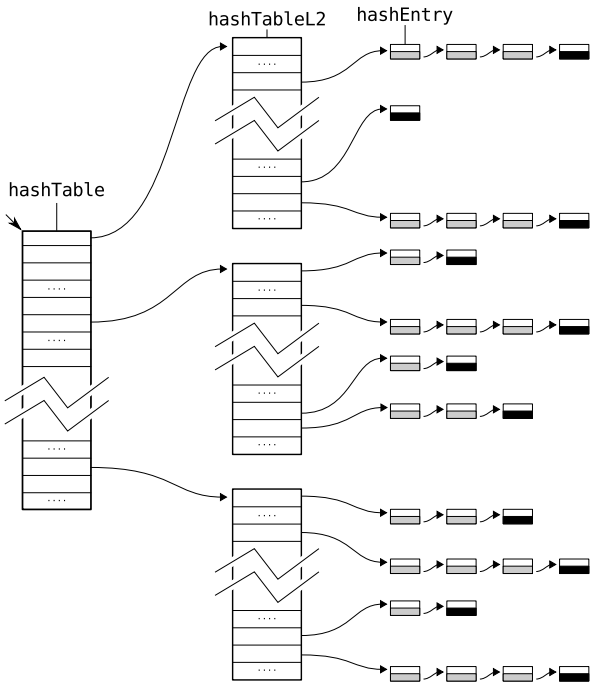
El valor de retorno de evalHash es un número de 16 bits, que debe interpretarse como dos valores diferentes; uno correspondiente a los primeros 8 bits y otro correspondiente a los segundos 8 bits.

La estructura de tabla de Hash es la siguiente:

```
typedef struct s_hashTable {
    hashTableL2* hashTable[256];
}__attribute__((packed)) hashTable;

typedef struct s_hashTableL2 {
    hashEntry* entry[256];
}__attribute__((packed)) hashTableL2;

typedef struct s_hashEntry {
    void*       elem;
    hashEntry*  prox;
}__attribute__((packed)) hashEntry;
```



Los primeros 8 bits del número devuelto por evalHash se utilizan como indice en la tabla inicial (hashTable). Esta entrada corresponde al puntero a la tabla de siguiente nivel (hashTableL2). Los segundos 8bits del número devuelto por evalHash se utilizan como indice en esta tabla para dar con la lista de elementos asociados a la clave.

La función eliminar (`void eliminar(hashTable* ht, void* clave, void* elem)`) se encarga de calcular la función de hash para la clave dada y obtener el índice correspondiente (haciendo el recorrido por los dos niveles de la tabla). Una vez encontrada la lista de elementos correspondientes a la clave, debe buscar y eliminar el elemento pedido.

1. (10 puntos) Escribir el pseudocódigo de la función pedida.
2. (30 puntos) Programar en ASM el código de la función anterior.

Ej. 3. (20 puntos)

Sea el siguiente código `ASM`,

```
f:
 movzx rcx, si
 shld rcx, 2
 add rdi, rcx
 push rdi
 movzx rdi, si
 cvtss2si rdx, xmm0
 add rdi, rdx
 mov rsi, rdi
 call g
 pop rdi
 mov [rdi], rax
 mov rax, [rax]
 ret
```

Donde las aridades de las funciones son:

- `int f(int** a, unsigned int short b, float C)`
- `int* g (int d, int e)`

1. (9 puntos) Explicar en palabras el funcionamiento y reescribir en lenguaje `C` el código anterior.
2. (11 puntos) Modificar el código `ASM` considerando que se modifican las aridades de las funciones como se muestra a continuación,

   - `int f(float x, int** a, unsigned int short b, float C)`
   - `int* g(float y, int* d, int* e, unsigned short z)`

   A la función `f` se le agrega un parámetro al principio denominado `x`. A la función `g` se le agrega un parámetro al principio denominado `y`, otro al final denominado `z`. Además los valores de los parametros `d` y `e` se deben tomar cómo referencia. Al llamar a la función `g`, el parámetro `y` debe tener el valor de la suma de `x` y `c`. Mientras que el parámetro `z` debe tener el valor `1955`.