Primer Parcial

Segundo Cuatrimestre 2022

Normas generales

- El parcial es INDIVIDUAL
- Una vez terminada la evaluación se deberá completar un formulario con el *hash* del *commit* del repositorio de entrega. El link al mismo es: `https://forms.gle/NGw8EvZD3KeAfXEw9`.
- Luego de la entrega habrá una instancia coloquial de defensa del trabajo

Régimen de Aprobación

- Para aprobar el examen es necesario obtener cómo mínimo **60 puntos**.
- Para conservar la posibilidad de promocionar es condición necesaria obtener como mínimo **80 puntos**.

Compilación y Testeo

El archivo `main.c` es para que ustedes realicen pruebas básicas de sus funciones. Sientanse a gusto de manejarlo como crean conveniente. Para compilar el código y poder correr las pruebas cortas implementadas en main deberá ejecutar `make main` y luego `./runMain.sh`.
En cambio, para compilar el código y correr las pruebas intensivas deberá ejecutar `./runTester.sh`. El programa puede correrse con `./runMain.sh` para verificar que no se pierde memoria ni se realizan accesos incorrectos a la misma.

Pruebas intensivas (Testing)

Entregamos también una serie de *tests* o pruebas intensivas para que pueda verificarse el buen funcionamiento del código de manera automática. Para correr el testing se debe ejecutar `./runTester.sh`, que compilará el *tester* y correrá todos los tests de la cátedra. Luego de cada test, el *script* comparará los archivos generados por su parcial con las soluciones correctas provistas por la cátedra. También será probada la correcta administración de la memoria dinámica.

Ej. 1 - (45 puntos)

Datos:

- Estructura `msg_t`:

Implementa un tipo de dato asociado a un mensaje etiquetado. Se encuentra definida en `ej1.h`. El mismo contiene un campo con el texto del mensaje, un campo con la longitud de dicho texto (**sin el caracter nulo**) y finalmente la etiqueta o *tag* del mensaje en cuestión representada por un número entre 0 y `MAX_TAGS-1`. En `C`:

```
typedef struct msg{
    char* text;
    size_t text_len;
    int tag;
} msg_t;
```

Finalmente, como restricción para la entrega, deberán incluir la solución completa en el archivo `ej1.asm`, es decir su solución debe estar totalmente contenida en este archivo.

Aunque se evaluará sólo la implementación en `asm`, se sugiere realizar previamente la implementación en C, en este caso, la misma deberá estar en `ej1.c`.

Para seleccionar qué implementación se testeará con `./runTester.sh` pueden modificar la constante `USE_ASM_IMPL` en el archivo `ej1.h`. Por ejemplo para testear la implementación en `asm` deberá estar definida:

```
#define USE_ASM_IMPL 1
```

Y para testear con la implementación en C (función `agrupar_c`), definir:

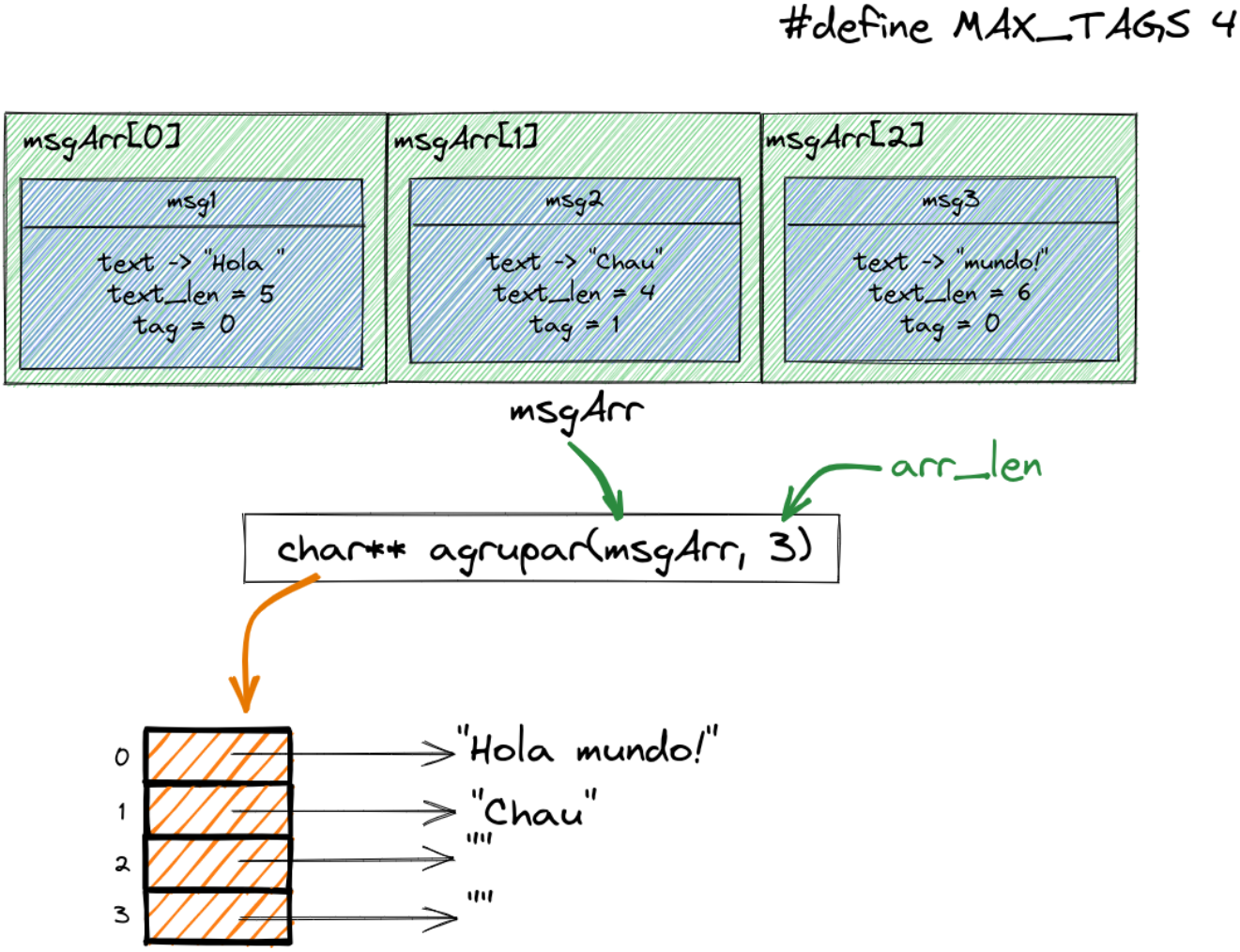
```
#define USE_ASM_IMPL 0
```

Se pide:

Implementar la siguiente función en `asm`:

```
char** agrupar(msg_t *msgArr, size_t msgArr_len);
```

La misma devuelve un arreglo de una longitud igual al número máximo de etiquetas definido, donde cada posición del array contiene un *string de C* con la concatenación del texto de todos los mensajes de la misma etiqueta o *tag*. Por ejemplo:



Ej. 2 - (30 puntos)

En este ejercicio van a programar un filtro que se aplica a un archivo de audio en formato WAV. Este es un formato PCM signed de 16 bits donde el canal izquierdo y el derecho vienen alternados de la siguiente manera:

R_0	L_0	R_1	L_1	R_2	L_2	R_3	L_3	...	R_n	L_n

El filtro tiene que cumplir lo siguiente para cada canal:

$$y_n = 1/4(x_n + x_{n+1} + x_{n+2} + x_{n+3}) \tag{1}$$

$$= 1/4(\sum_{i=n}^{n+3} x_i), \tag{2}$$

donde:

- $x[n]$ es la señal de entrada en la posición n y para cada canal

- $y[n]$ es la señal de salida en la posición n y para cada canal

O de manera gráfica:

$1/4(\sum_{i=0}^{3} R_i)$	$1/4(\sum_{i=0}^{3} L_i)$	$1/4(\sum_{i=1}^{4} R_i)$	$1/4(\sum_{i=1}^{4} L_i)$	$\cdots$	$1/4(\sum_{i=n-3}^{n} R_i)$	$1/4(\sum_{i=n-3}^{n} L_i)$

Notar que:

- En caso de que el resultado no sea entero, este debe ser truncado.

- Además si N es la longitud de la entrada en cantidad de datos (1 dato = 4 bytes teniendo en cuenta ambos canales), siempre se va a cumplir que N=4*k+3, siendo k un entero mayor a cero

- Este ejercicio debe realizarse utilizando el paradigma de programación SIMD, en caso contrario será considerado incorrecto.

Preguntas teóricas - (25 puntos)

Tema 2

a ¿Cual es el efecto que produce en el rendimiento de un sistema x86, almacenar el número 0xFFEEDDCC a partir de la dirección 0x16000003?. ¿Cuál es la solución? ¿Qué costo implica esa solución?

b Dados dos registros SIMD que tienen los siguientes valores:

A07C	D985	DE09	3788
71CC	0768	3259	98E0

indicar cuánto valen las sumas con aritmética de desborde y con aritmética saturada con y sin signo si cada paquete es de 16 bits.