

## Primer Parcial

### Segundo Cuatrimestre 2023

### 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/QbL5Z879To9JEJt48>.
- 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 - (50 puntos)

La famosa fintech *orga2-pay* maneja miles de pagos por día. Como en toda fintech, el fraude y el robo de cuentas es un problema que genera importantes pérdidas económicas. Si le roban la cuenta a una empresa se pueden robar miles y miles de dólares. Para prevenir esto, *orga2-pay* tiene un equipo de revisión manual que manda pagos a revisar cuando parecen provenir de una cuenta robada, para evitar que hackers se lleven todo el dinero de ella. En esta ocasión nos contrataron para armar un sistema que permita priorizar ciertos casos sobre otros, y por cuestiones de velocidad nos pidieron que sea escrito en ASM.

Nuestros pagos tienen la siguiente forma:

```
typedef struct {
    uint8_t monto;
    uint8_t aprobado;
    char* pagador;
    char* cobrador;
} pago_t;
```

Para los sistemas de machine learning de prevención de fraude es importante diferenciar los pagos aprobados de los rechazados de un usuario dado. Para esto tenemos el siguiente struct:

```
typedef struct {
    uint8_t cant_aprobados;
    uint8_t cant_rechazados;
    pago_t** aprobados;
    pago_t** rechazados;
} pagoSplitted_t;
```

## Se pide:

Implementar en asm:

- a Dos funciones que cuenten la cantidad de pagos aprobados y rechazados de un usuario dado. La aridad de las funciones es:

- `uint8_t` contar\_pagos\_rechazados\_asm(list\_t\* pList, `char*` usuario)
- `uint8_t` contar\_pagos\_aprobados\_asm(list\_t\* pList, `char*` usuario)

- b La función `split_pagos_usuario_asm`, que toma una lista enlazada de pagos y un usuario, y devuelve un puntero a `pagoSplitted_t` donde están los pagos rechazados y aprobados que el usuario dado recibió. Es decir, los que lo tienen como **cobrador**. Su aridad es:

```
pagoSplitted_t* split_pagos_usuario_asm(list_t* pList, char* usuario);
```

Tener en cuenta lo siguiente:

- Se puede usar `strcmp` para realizar comparaciones entre strings
- No se deben hacer copias de los pagos. El struct `pagoSplitted_t` contiene arrays de punteros

**Consejo:** Utilizar las funciones del punto 'a' en la implementación del 'b'.

## Ej. 2 - (50 puntos)

En Orga2 llamamos `mezclarColores` al filtro que dada una imagen en color (con formato RGBA) nos devuelve otra imagen del mismo tamaño (también con formato RGBA) tal que en cada uno de sus píxeles presenta un reordenamiento de los colores del píxel en la misma posición de la imagen original. Cómo se reordenarán los colores dependerá de la relación entre los valores de los colores de la imagen original.

Más concretamente, siendo  $X$  la imagen original e  $Y$  la imagen resultante, `mezclarColores` se define de la siguiente forma:

$$\text{mezclarColores}(X[ij]) = \begin{cases} Y[ij]_R = X[ij]_B, Y[ij]_G = X[ij]_R, Y[ij]_B = X[ij]_G & \text{si } X[ij]_R > X[ij]_G > X[ij]_B \\ Y[ij]_R = X[ij]_G, Y[ij]_G = X[ij]_B, Y[ij]_B = X[ij]_R & \text{si } X[ij]_R < X[ij]_G < X[ij]_B \\ Y[ij] = X[ij] & \text{si no} \end{cases}$$

El valor de la componente de transparencia en el resultado debe ser 0 para todos los píxeles.

```
void mezclarColores(uint8_t *X, uint8_t *Y, uint32_t width, uint32_t height)
```

Notar que:

- Se recibe un puntero al espacio de memoria donde debe guardarse el resultado, es decir, esa memoria ya fue pedida (o reservada)