Primer Parcial

Primer 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/tcAEEMwsREtPNQNF6` (se encuentra en el campus también)).
- 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)

Los templos griegos del periodo clásico suelen ser de planta rectangular rodeados de columnas. Estos respetan la siguiente regla sobre la cantidad de columnas de sus lados: 2N+1=M, siendo M el lado largo y N el lado corto.

Se tiene un arreglo de templos que almacena en cada posicion las características de un templo, nombre y cantidad de columnas de cada lado (ver struct abajo).

```
typedef struct {
  uint8_t colum_largo;
  char *nombre;
  uint8_t colum_corto;
} templo;
```

Se pide:

Implementar las siguientes funciones en `asm`:

```
uint32_t cuantosTemplosClasicos(templo *temploArr, size_t temploArr_len);
```

La misma devuelve la cantidad de templos que son del periodo clasico, es decir cumplen con la condicion mecionada para la cantidad de columnas.

```
templo* templosClasicos(templo *temploArr, size_t temploArr_len);
```

La misma recibe un arreglo de templos y devuelve un nuevo arreglo que solo contiene los templos del arreglo original que son del periodo clasico.

Ej. 2 - (50 puntos)

En Orga2 llamamos `miraQueCoincidencia` al filtro que dadas dos imágenes en color (con formato RGBA) de tamaño NxN nos devuelve otra de del mismo tamaño pero en escala de grises, llamada `laCoincidencia`, donde sólo se encuentran aquellos pixeles (transformados a escala de grises) que coinciden en ambas imágenes, los que no coinciden son puestos en blanco. Para ser más concretos la podemos definir de la siguiente forma:

$$miraQueCoincidencia[ij] = \begin{cases} convertirEscalaGrises(A[ij]) & \text{si } A[ij] = B[ij] \\ 255 & \text{si no} \end{cases}$$

Tener en cuenta que para convertir a escala de grises cada pixel es representado por un unico byte que se obtiene a partir de operar sobre el pixel de la imagen a color de la siguiente manera: 0.299 * Rojo (R) + 0.587 * Verde (G) + 0.114 * Azul (B). Notar que el byte de la transparencia (A) será ignorado.

Se pide:

Implementar la siguiente función en `asm` usando instrucciones de SIMD:

```
void miraQueCoincidencia( uint8_t *A, uint8_t *B, uint32_t N,
                          uint8_t *laCoincidencia )
```

Notar que:

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

- Alcanza operar con floats (no es necesario usar doubles)