

Recuperatorio 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/STtMNBwai6PjmyBw6> (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**.

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 - (55 puntos)

El objetivo de este ejercicio es implementar un conjunto de funciones sobre una lista doblemente enlazada de tamaño variable. Para ello, tenemos una estructura que indica el primer y el último nodo, y cada nodo tiene punteros al anterior y al siguiente. Además, cada nodo tiene un tipo y un hash.

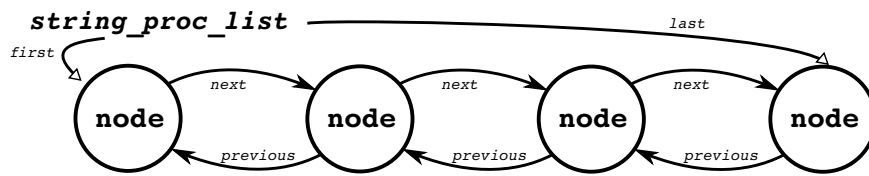


Figura 1: Ejemplo de Lista Doblemente Enlazada.

Las estructuras tienen la siguiente definición:

```

/** Lista */
typedef struct string_proc_list_t {
    struct string_proc_node_t* first;
    struct string_proc_node_t* last;
} string_proc_list;

/** Nodo */
typedef struct string_proc_node_t {
    struct string_proc_node_t* next;
    struct string_proc_node_t* previous;
    uint8_t type;
    char* hash;
} string_proc_node;

```

Se pide implementar las siguientes funciones:

1. `string_proc_list* string_proc_list_create_asm(void);`

Inicializa una estructura de lista.

2. `string_proc_node* string_proc_node_create_asm(uint8_t type, char* hash);`

Inicializa una estructura de nodo con el tipo y el hash dado.

El nodo tiene que apuntar al hash pasado por parámetro (no hay que copiarlo).

3. `void string_proc_list_add_node_asm(string_proc_list* list, uint8_t type, char* hash);`

Dada una lista, un tipo y un hash, agrega un nodo nuevo al final de la lista con el tipo y el hash dado.

Al igual que en el anterior, no hay que copiar el hash. Debe apuntar al mismo.

4. `char* string_proc_list_concat_asm(string_proc_list* list, uint8_t type, char* hash);`

Genera un nuevo hash concatenando el pasado por parámetro con todos los hashes de los nodos de la lista cuyos tipos coinciden con el pasado por parámetro.

Ej. 2 - (45 puntos)

En Orga2 llamamos `combinarImagenes` al filtro que das dos imágenes (con formato BGRA) de igual tamaño, devuelve una tercera que combina las dos de la siguiente forma:

$$\begin{aligned} res[ij]_B &= A[ij]_B + B[ij]_R \\ res[ij]_G &= \begin{cases} A[ij]_G - B[ij]_G & \text{si } A[ij]_G > B[ij]_G \\ promedio(A[ij]_G, B[ij]_G) & \text{si no} \end{cases} \\ res[ij]_R &= B[ij]_B - A[ij]_R \end{aligned}$$

El valor de la componente de transparencia en el resultado debe ser 255 para todos los píxeles. Implementar el filtro de manera que procese de al menos **dos píxeles** simultáneamente.

```
void combinarImagenes(uint8_t *src1, uint8_t *src2, uint8_t *dst,
                     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 alocada)
- Para el promedio se puede utilizar la instrucción `pavgb/pavgw`, o calcularlo siguiendo la fórmula
$$\frac{(A[ij]_G + B[ij]_G + 1)}{2}$$