Ejercicio Repaso Primer Parcial

Segundo Cuatrimestre 2023

El siguiente ejercicio formó parte del Recuperatorio del Primer Parcial correspondiente al Primer Cuatrimestre de 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/CkonqkTcoJNF7mAu8` (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 - (50 puntos)

Una importante fintech llamada Orga2-Libre nos contrató para mejorar su sistema de prevención de fraude en los pagos realizados con su tarjeta prepaga. Para esto, nos dieron un conjunto de datos de pagos de 10 clientes, como todavía no saben sin van a contratarnos no quieren darnos información adicional. Un pago tiene la siguiente forma:

```
typedef struct {
        uint8_t monto;
        char *comercio;
        uint8_t cliente;
        uint8_t aprobado;
} pago_t;
```

donde `monto` es el monto del pago, `comercio` el nombre del comercio, `cliente` es el numero de cliente (en esta primera versión sólo serán números del 0 al 9) y `aprobado` es un booleano que nos dice si el pago fue aprobado o no.

Se pide:

1. **(20 puntos)** Se desea conocer el monto total de pagos realizados por cliente.

   Para esto, queremos construir una función en ASM que tome un arreglo de pagos y nos devuelva un arreglo de 10 posiciones, donde en cada posición esté el monto total de pagos aprobados por cada cliente.

   Si un cliente no realizó ningún pago, se devolverá 0 en la posición correspondiente.

   ```
   uint32_t* acumuladoPorCliente_asm(uint8_t cantidadDePagos, pago_t* arr_pagos);
   ```

2. **(30 puntos)** Muchas veces desde Orga2-Libre quieren bloquear comercios. Para esto es importante saber cuantos pagos se rechazarán en caso de bloquear un comercio.

   Se pide realizar:

   (a) una función `en_blacklist_asm` que dado el nombre de un comercio, una lista de nombres de comercios y la longitud de dicha lista, retorne `1` si el comercio esta en esa lista y `0` en caso contrario.

   ```
   uint8_t en_blacklist_asm(char* comercio, char** lista_comercios, uint8_t n);
   ```

   (b) Una función `blacklistComercios_asm` que dado:

   - `cantidad_pagos` una cantidad de pagos
   - `arr_pagos` un arreglo de pagos,
   - `arr_comercios` un arreglo de nombres de comercios a incluir en la *blacklist*,
   - `size_comercios` el tamaño del arreglo de nombres de comercios

   devuelva un arreglo de punteros a todos los pagos que hayan sido realizados en alguno de los comercios de la lista `arr_comercios`.

   ```
   pago_t** blacklistComercios_asm(uint8_t cantidad_pagos,
                                   pago_t* arr_pagos,
                                   char** arr_comercios,
                                   uint8_t size_comercios);
   ```