Recuperatorio de Primer Parcial

Primer 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/Z5yHKPP7giNSHhqM6`.
- Luego de la entrega habrá una instancia coloquial de defensa del trabajo

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.

Enunciado

En esta oportunidad, vamos a trabajar con la estructura:

Estructura `letters_quantity`

```
typedef struct letters_quantity {
uint8_t consonants_qty;
char* word;
uint8_t vowels_qty;
} letters_quantity_t;
```

Que guarda una palabra (word) con su correspondiente cantidad de vocales (vowels_qty) y consonantes (consonants_qty).

Y se han definido las siguientes funciones:

- `uint8_t countVowels(char* word)`
  Dada una palabra retorna la cantidad de vocales
- `letters_quantity_t* createLettersQuantityArray(uint8_t size)`
  Devuelve un arreglo de tamaño size de letters_quantity.
- `void addToArray(char* word, letters_quantity_t wq_array[], uint8_t position))`
  Agrega al arreglo la palabra word en una estructura de letters_quantity en la posición del arreglo indicada por position. La función sólo esta definida cuando position es menor que el tamaño del arreglo.
- `char* getMaxVowels(letters_quantity_t* wq_array, uint8_t array_size)`
  Recorre el arreglo y devuelve la palabra con más vocales
- `void cleanWQArray(letters_quantity_t wq_array[], uint8_t size)`
  Libera la memoria tomada por el arreglo wq_array.
- `void arrayPrint(letters_quantity_t* wq_array, uint8_t size, FILE* pFile)`
  Imprimer un arreglo de letters_quantity

Primero, responda las siguientes preguntas provistas en el archivo respuestas.txt.

1. Observe la estructura provista *letters_quantity*: ¿Cuántos bytes ocupa en memoria? ¿Cómo podría usar C para averigüarlo?
2. ¿Cuál es el desplazamiento de cada variable en la estructura en memoria y que tamaño en bytes tiene cada variable?
3. ¿Cómo reorganizaría las variables de la estructura *letters_quantity* para ahorrar memoria?
4. ¿Qué hace la función *cleanWQArray*? Explique brevemente el uso de free y malloc.

Les damos todas las funciones implementadas en C, pero les vamos a pedir que implementen algunas en Assember.

5. Implemente en Assembler la función *countVowels* Ejemplos:
   countVowels("astronomia") = 5
   countVowels("orga2") = 2
   countVowels("boca") = 2
   countVowels("clk") = 0

   - Asuma que todas las letras son minúsculas
   - Los valores hexadecimales de las vocales en ASCII son a: 61, e: 65, i: 69, o: 6F, u:75
6. Implemente en Assembler la función *createLettersQuantityArray*
7. Implemente en Assembler la función *getMaxVowels*. Para esta función no se puede llamar a countVowels, tiene que trabajar con la información que se encuentra en los structs del arreglo.