

Semestre 2020 - 1

Profesor P1: Fernando Auat

Profesor P2: Ioannis Vourkas

Contenidos principales

- Llamado a funciones y uso de archivos de texto de I/O
- Uso de punteros y manejo de memoria dinámica
- Estructuras de datos: Lista enlazada, STACK, y Arbol Binario de Búsqueda (ABB)

Objetivos

- Familiarizarse con el desarrollo modular/incremental de código C
- Practicar con las estructuras de datos: Lista enlazada, STACK, y Arbol Binario de Búsqueda (ABB)

IMPORTANTE: Las tareas son individuales. Cualquier acción que pueda beneficiar de forma injusta la calificación de su tarea está prohibida, incluyendo la presentación de cualquier componente que no es de su autoría, o la facilitación de esto para otros. Es aceptable *discutir -en líneas generales- los métodos y resultados con sus compañeros*, pero **se prohíbe compartir soluciones de código. Utilizar código de internet que no es de su autoría, también es considerado plagio, a menos que se indique la fuente.** **Presten atención a las instrucciones de entrega**, que pueden incluir políticas de nombre de archivos y aspectos que se considerarán en la evaluación.

Descripción del trabajo a realizar

Parte A: *Preparación de datos de Entrada* (25 Pts)

Se entrega una lista de 25 alumnos (todos famosos) de un curso de EDA hipotético según su nota en el primer certamen. Los datos se encuentran en el archivo `notas-EDA-C1.txt`. En dicho archivo, cada línea tiene un solo string correspondiente a un alumno, conformado por 3 campos separados por una coma ',', de la siguiente forma:

Nombre,Apellido,nota

Su programa debe leer la información almacenada en el archivo `notas-EDA-C1.txt`, cuyo nombre **se pasa como parámetro vía la línea de comandos**. Para este propósito, se recomienda utilizar la función:

```
char * strtok(char * str, const char * delimiters)
```

(disponible en la biblioteca string.h) de manera repetitiva, para facilitar la identificación de los tres campos en cada línea del archivo.

Luego su programa debe encriptar el nombre y apellido de cada alumno, aplicando un algoritmo de criptografía conocido como el “**cuadrado de Polibio**”, llamando la función:

```
char* encrypt(char* str)
```

La encriptación se realizará según el **cuadrado de Polibio**. Inventado hacia 150 a.C. por el historiador griego Polibio, el cuadrado de Polibio consiste en un algoritmo trivial de criptografía donde **cada letra del alfabeto es reemplazada por las coordenadas de su posición en un cuadrado**. Tomemos como ejemplo un cuadrado de Polibio en la **Figura 1**.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I, J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Fig. 1 Cuadrado de Polibio

En este caso, hemos puesto la I y la J juntas para poder entrar en 25 celdas. La codificación consiste simplemente en indicar la fila y columna que ocupa cada letra, de forma sucesiva, en el cuadrado. Por ejemplo, la *w* está en la línea 5 y en la columna 2, y por lo tanto le corresponde el 52. Por ejemplo, el texto «Wikipedia, la enciclopedia libre» se codificará así:

```
52 24 25 24 35 15 14 24 11 31 11 15 33 13 24 13 31 34 35 15 14 24 11 31 24 12 42 15
w i k i p e d i a l a e n c i c l o p e d i a l i b r e
```

PALTA: Para la función `char* encrypt(char*)`, pueden tener un arreglo bidimensional `char polybius_quadrant[][]` y recorrerlo buscando cada vez la letra que quieren encriptar, y cuando la encuentren, se quedan con los índices de la celda donde se encuentra la letra (y les suman 1).

Parte B: *Uso de STACK y Listas Enlazadas* (40 Pts)

A medida que se vayan leyendo los datos del archivo, crearán un nuevo nodo para cada estudiante según la siguiente estructura, donde nombre y apellido serán ambos encriptados:

```
#define NLEN 30

typedef struct nd {
    char nombre[NLEN];
    char apellido[NLEN];
    int nota;
    struct nd *previous; //equivalente a left en ABB
    struct nd *next;     //equivalente a right en ABB
}Nodo;
```

Cada nodo será agregado a un **STACK** implementado con una lista doblemente enlazada. La interfaz para el manejo del STACK debe hacerse en los archivos **myStack.h** y **myStack.c**, que deben tener una estructura struct con las variables necesarias para administrar el Stack, e incluir como mínimo las funciones que se presentan (en forma genérica) a continuación:

```
isFull()           // indica si el stack está lleno
isEmpty()          // indica si el stack está vacío
getNewStack()      // genera un nuevo stack
stackDelete()      // elimina el stack y libera la memoria reservada
push()             // agrega un elemento al stack
pop()              // quita el primero elemento del stack
stackPrint()       // imprime los contenidos del stack completo
```

Al completar esta parte y tras formar el STACK, su programa debe imprimir los contenidos por pantalla.

Parte C: *Uso de Árbol Binario de Búsqueda* (35 Pts)

En esta última parte, su programa debe quitar los datos del STACK antes formado e **ingresarlos a un Árbol Binario de Búsqueda (ABB)** según el valor de su nota. El ABB es una estructura de datos

fundamental en ciencias de la computación, muy efectiva para almacenar datos ordenados y también para recuperar rápidamente datos almacenados de manera ordenada. Asegúrense de que su ABB acepte notas repetidas.

En el contexto del ABB, en la estructura **Nodo** puede considerar el campo `previous` como `left`, y el campo `next` como `right`. La interfaz para el manejo del ABB debe hacerse en los archivos **myABB.h** y **myABB.c**, que implementarán todas las funciones necesarias, incluidas las que son mencionadas a continuación (en forma genérica).

La inserción de nuevos nodos al ABB se realizará mediante la función `insertNode()`. Una vez creado el árbol, su programa debe recorrer el árbol y [mostrar su contenido por pantalla de mayor a menor nota y de menor a mayor](#), ejecutando las funciones "recursivas" `showABBMaxMin()` y `showABBMinMax()`.

Al completar este paso, su programa [debe eliminar el ABB](#) ejecutando la función `destroyABB()`, y [también eliminar el STACK](#) anteriormente creado, antes de terminar la ejecución.

Consideraciones y Formato de entrega

- Todas las funciones que NO forman parte de la interfaz del Stack y del ABB, serán definidas en el archivo `main.c` en la parte inferior, después de la función `main()`.
- Utilice ***/*comentarios*/*** para documentar lo que se hace en cada etapa de su código. **Se debe además usar el estilo de documentación Doxygen** utilizado en TODOS los ejemplos de código que encontrarán en AULA.
- El código deberá estar perfectamente *indentado con espacios*, no tabuladores
- Toda tarea debe ser correctamente **compilada** y ejecutada **en el servidor Aragorn** (tareas que **no compilan** se evaluarán cualitativamente y **tendrán como máximo nota 54**).
- Para la entrega, suba **un archivo** comprimido (.zip, .rar, .tar.gz, etc.) que contenga su código (es decir, **SOLAMENTE** sus archivos **.c** y **.h**), más un archivo `README.txt`, donde especificará sus datos personales (**nombre, apellido, ROL**) y, de ser necesario, explicará cualquier particularidad que pueda tener su programa, como por ejemplo cómo debe ser compilado y ejecutado.

¿Cómo ENTREGAR?

El entregable debe ser subido a la plataforma AULA, **hasta las 23:50 del viernes 19 de junio, 2020**.