Estructuras de Datos y Algoritmos - ELO-320

Primer semestre 2020 - Campus San Joaquín

Francisco Cabezas
francisco.cabezas@xkivertech.cl

---

1. Tarea 1

Contenidos principales

- Llamado a funciones y lectura de archivos de texto
- Uso de punteros y manejo de memoria dinámica
- Estructuras de datos: Lista enlazada, STACK y Árbol Binario de Búsqueda (ABB)

Objetivos

- Familiarizarse con el desarrollo modular e incremental de código C
- Practicar con las estructuras de datos: Lista enlazada, STACK y Árbol 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 certámen. 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 línea de comandos**. Para este propósito, se recomienda usar 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 a 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 como en la siguiente figura:

	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

En este caso, las letras I y J están juntas para calzar las 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 letra W está en la fila 5 y en la columna 2, por tanto le corresponde el 52**. Por ejemplo, el texto `Colectiva` se codificará así:

C	O	L	E	C	T	I	V	A
13	34	31	15	13	44	24	51	11

Hints: No se consideran los espacios, tampoco hace distinción entre mayúscula y minúscula. Tampoco considera la ñ.
Para la función `char* encrypt (char*)`, puede 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 le suman 1).

Parte B: *Uso de STACK y Listas enlazadas (40 pts.)*

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

```c
#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 con las variables necesarias para administrar el STACK, e incluir como mínimo las funciones que se presentan (en forma genérica) a continuación:

```c
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 nuevo elemento al stack

pop();         // Quita el primer 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 (40 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. Asegúrese 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, incluídas las que son mencionadas a continuación (en forma genérica):

```c
insertNode();     // Permite insertar nuevos nodos al ABB

showABBMaxMin(); // Muestra el contenido ordenado por notas, de mayor a menor

showABBMinMax(); // Muestra el contenido ordenado por notas, de menor a mayor

destroyABB();     // Elimina el árbol binario
```

Una vez creado el árbol, su programa debe recorrer el árbol y mostrar su contenido por pantalla, primero con las notas de mayor a menor, y luego de menor a mayor.

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. Es vital documentar cada método para entender qué hace, qué parametros recibe y cual es su salida.
- El código deberá estar *identado* y ordenado.
- Toda tarea debe ser correctamente compilada y ejecutada. Aquellas tareas que no compilen serán evaluadas cualitativamente y **tendrán como máximo nota 54** (**Procure que compile!!!**).
- 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, una breve explicación de lo que logró hacer en su programa, e instructivo para compilar.

Hint

Hasta el momento en las clase hemos visto como consultar funciones que están implementadas en el mismo archivo `.c`, pero es posible separar la funcionalidad en distintos archivos para así cumplir con el principio de delegación de responsabilidades y mejorar la legibilidad de nuestro código. El siguiente link es altamente recomendado para entender cómo hacerlo.

¿Cómo entregar?

Su entregable debe ser subido a la plataforma AULA, **hasta las 23:59 del Martes 23 de Junio, 2020**.