TRABAJO PRÁCTICO 3:
LAS FAUCES DE LA PERDICION

75.40 Algoritmos y Programación I
Curso: Méndez

5 de noviembre de 2017

1. Objetivo

- Que alumnos y alumnas puedan analizar un problema y elaborar una solución aplicando los conceptos de algoritmia y programación relacionados con manejo de archivos.
- Que alumnos y alumnas demuestren que manejan las prácticas de buena programación.

2. Historia

Gracias al increíble operativo de preparación de las naves de guerra y habiendo realizado miles de simulaciones para detectar la mejor estrategia, la Estrella de la Muerte fue finalmente destruida por el torpedo de una simple nave atada con alambre.
El espacio se iluminó brevemente ante semejante explosión para que al disiparse la ceguera temporal se revelara algo aún peor: la Estrella de la Muerte era una distracción que le daba tiempo a otra nave a unirse, y terminar, la batalla, que recibió el nombre de Fauces de la Perdición. Esta increíble nave tenía facultades autorreparadoras y autodefensivas, significando realmente la perdición de la galaxia.
Ante semejante amenaza, los escuadrones de la Alianza Rebelde se debieron reorganizar rápidamente. Inteligencia detectó que Fauces tenía varios puntos débiles que no se autorreparan pero que obviamente estarían altamente custodiados.
La tarea de armar una buena estrategia recae una vez más en nosotros... ¿seremos capaces de aceptarla y vencer de una vez por todas?

3. Especificaciones funcionales

Se debe desarrollar una biblioteca que ponga a disposición una serie de funciones que permitan desarrollar la batalla con Fauces de la Perdición utilizando distintos archivos binarios y de texto. Tener en cuenta que el usuario va a ser otro programador que va a consumir nuestra biblioteca, por lo cual es importante que nuestra implementación **no tenga ningún tipo de interacción con el usuario, ni por teclado ni por pantalla**.
Cada una de las funciones requeridas **no tendrán precondiciones ya que todos los parámetro deben ser validados**. Ante la recepción de parámetros inválidos las funciones deben devolver false y en el caso contrario se procederá con el accionar esperado de las funciones y devolver true. En este sentido, las poscondiciones deben dar cuenta de esos casos, por ejemplo: *La función devuelve false en los casos X, Y y Z, caso contrario hace cierta cosa y devuelve true.*
Una validación general para todas las funciones es que los archivos involucrados deben poder abrirse, ya sean para lectura o escritura según se necesite. En el caso de no poder realizar esta operación se debe abortar la ejecución de la función, es decir cerrar todos los archivos que hayan sido previamente abiertos y devolver false, sin ejecutar nada del resto de la funcionalidad.
Fauces de la Perdición posee, por ahora, hasta 7 vulnerabilidades que cada una tiene una debilidad a un elemento químico distinto. Las vulnerabilidades encontradas son a los siguientes elementos: hidrógeno, yodo, carbono, nitrógeno, oxígeno, azufre y fósforo. Sabiendo eso, la Alianza Rebelde desarrolló torpedos cuyo componente

principal es alguno de estos elementos. Luego distribuyó los torpedos de dichos elementos a los distintos escuadrones, teniendo en cuenta que todas las naves de un escuadrón tienen el mismo elemento de ataque.
Para armar la estrategia de ataque primero se cargará la información correspondiente a los escuadrones y las vulnerabilidades, utilizando funciones provistas para tales fines. Luego se realizará la distribución de los escuadrones que atacarán a las vulnerabilidades y con el archivo obtenido de esa función Inteligencia generará la estregia de ataque para saber si al final Fauces de la Perdición será destruida o no.

4. Especificaciones de los archivos

- **Maestro de escuadrones: SQUADS.dat**
  Archivo binario donde cada registro es un struct que contiene: código de escuadrón (char), cantidad de naves del escuadrón (int), elemento de ataque (char), potencia de los ataques (int). No está necesariamente ordenado pero se sabe que a lo sumo habrá 26 registros.

SQUADS.DAT

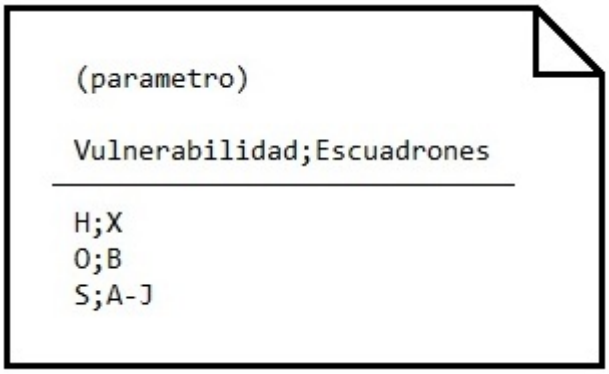
Código	Cantidad	Elemento	Potencia
A	20	S	20
B	10	O	30
X	30	H	5
J	5	S	40

- **Maestro de vulnerabilidades: VULNERA.dat**
  Archivo binario donde cada registro es un struct que contiene: elemento al que es débil (char, campo código), resistencia (int), durabilidad (int). Está ordenado ascendentemente por código de elemento y a lo sumo tiene 7 registros (pues son 7 las vulnerabilidades conocidas hasta ahora).

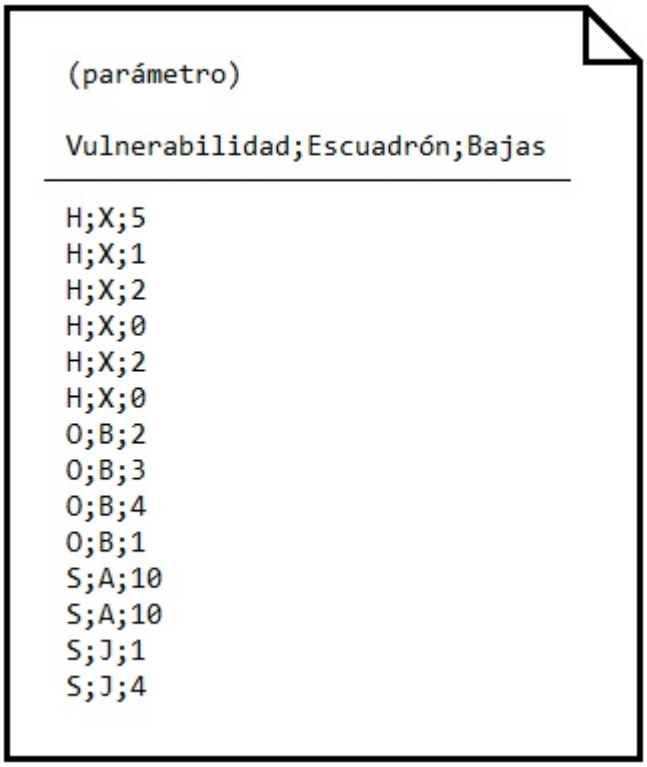
VULNERA.DAT

Elemento	Resistencia	Durabilidad
H	400	200
O	500	500
S	10000	10

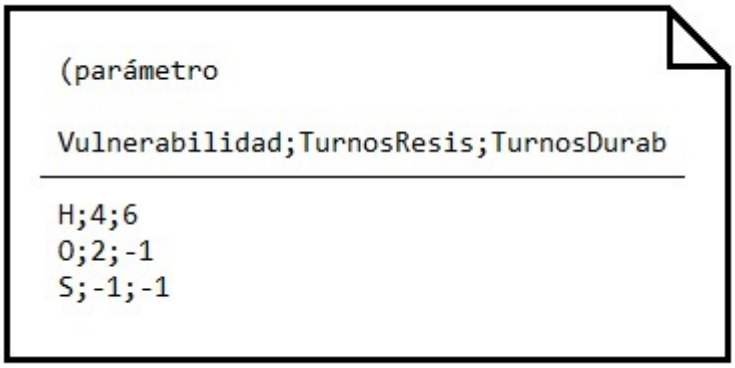
- **Distribución de escuadrones por vulnerabilidad: (parámetro)**
  Archivo de texto donde cada línea contiene: código de vulnerabilidad, códigos de escuadrón separados por guiones en el medio (Por ejemplo: “A-B-C”). Ambos campos se encuentran separados por punto y coma (;). Está ordenado ascendentemente por código de vulnerabilidad y posee tantas líneas como vulnerabilidades haya cargadas en el archivo maestro.



- **Ataques por vulnerabilidad y escuadrón: (parámetro)**
  Archivo de texto donde cada línea contiene: código de vulnerabilidad, código de escuadrón, bajas del escuadrón (número entero). Todos los campos se encuentran separados por punto y coma (;). Está ordenado ascendentemente por código de vulnerabilidad y código de escuadrón pero se desconoce la cantidad de líneas que tiene.



- **Resumen de ataques: (parámetro)**
  Archivo de texto donde cada línea contiene: código de vulnerabilidad, cantidad de ataques hasta que la resistencia llegó a 0 (número entero), cantidad de ataques hasta que la durabilidad llegó a 0 (número entero). Todos los campos se encuentran separados por punto y coma (;). Está ordenado ascendentemente por código de vulnerabilidad y posee tantas líneas como vulnerabilidades haya.



Nota: La estructura de cada registro/línea de los archivos será siempre válida, es decir cada archivo tendrá la cantidad de campos que se indica y del tipo de dato indicado. No se necesita validar que haya menos o más campos ni que sean de un tipo de dato distinto al explicitado en el enunciado.

5. Especificaciones de implementación

Se debe desarrollar la biblioteca fauces (.h y .c) que debe contar con las siguientes funciones:

- **bool actualizar_escuadrones(char codigo_escuadron, int cantidad_naves, char elemento_ataque, int potencia_ataque)**
  A cualquier validación se le debe agregar que el código de escuadrón debe ser una letra mayúscula, la cantidad de naves y la potencia de ataque deben ser siempre positivas y que el elemento de sus ataques tiene que ser uno de los preestablecidos.
  Si el escuadrón no existe se lo dará de alta en el archivo maestro, pero si existe se actualizarán los datos con los pasados por parámetro.

- **bool actualizar_vulnerabilidades(char* archivo_actualizaciones)**
  Se darán de alta los registros con las vulnerabilidades que no existan en el archivo maestro, pero las que existan serán actualizadas con los datos del registro del archivo de actualizaciones teniendo que en cuenta que ésto se hará en el único caso de que el registro de actualizaciones sea válido (que los puntos de resistencia y de durabilidad sean positivos y que el elemento al que es débil sea uno de los preestablecidos). En este último caso, si un registro de actualizaciones no es válido simplemente se ignorará y se continuará con los restantes.

- **bool distribuir_escuadrones(char* archivo_distribuciones)**
  Generará el archivo de distribuciones de escuadrones por vulnerabilidad. Por cada vulnerabilidad en el maestro se obtendrán los escuadrones cuyo elemento de ataque coincida con dicha la debilidad de dicha vulnerabilidad.

- **bool realizar_ataques(char* archivo_ataques, char* archivo_resumen, bool* fauces_destruida)**
  El archivo de ataques tiene lotes de registros por vulnerabilidad, y a su vez en esos lotes hay sublotes que corresponden a los escuadrones atacantes. Al inicio de un lote de ataques por vulnerabilidad se deben recuperar del archivo maestro de vulnerabilidades sus datos (resistencia y durabilidad); al inicio de un sublote de ataques por escuadron se deben recuperar sus datos (cantidad de naves y potencia de ataque de cada nave).
  La dinámica de esta función es la siguiente: por cada registro se multiplicará la cantidad de naves del escuadrón actual por la potencia de ataque de las naves de dicho escuadrón y se le restará a la resistencia de la vulnerabilidad hasta que la misma sea 0; cuando la resistencia llega a 0 el proceso será el mismo pero para la durabilidad hasta que también llegue a 0. Luego de cada ataque se debe ir descontando a la cantidad de naves del escuadrón las naves eliminadas. El objetivo es tratar de destruir cada una de las vulnerabilidades, es decir que su durabilidad llegue a 0. Este proceso se debe repetir para cada una de las vulnerabilidades.
  Al finalizar todos los ataques se debe generar el archivo de resumen de ataques. Tener en cuenta que si

alguna de las cantidades de ataques nunca llegó a 0 entonces dicha cantidad será reemplazada por -1 para indicar que no se logró llegar a 0 ahí.

Línea de ataque	Vulnerabilidad			Escuadrón			
	Código	Resistencia	Durabilidad	Código	Cantidad	Potencia	Bajas
H;X;5	H	400	200	X			5
	H	400	200	X	30	5	5
	H	400 - 30*5 = 250	200	X	30 - 5 = 25	5	5
H;X;1	H	250 - 25*5 = 125	200	X	25 - 1 = 24	5	1
H;X;2	H	125 - 24*5 = 5	200	X	24 - 2 = 22	5	2
H;X;0	H	5 - 22*5 = 0	200	X	22 - 0 = 22	5	0
H;X;2	H	0	200 - 22*5 = 90	X	22 - 2 = 20	5	2
H;X;0	H	0	90 - 20*5 = 0	X	20 - 0 = 20	5	0
O;B;2	O	500	500	B			2
	O	500	500	B	10	30	2
	O	500 - 10*30 = 200	500	B	10 - 2 = 8	30	2
O;B;3	O	200 - 8*30 = 0	500	B	8 - 3 = 5	30	3
O;B;4	O	0	500 - 5*30 = 350	B	5 - 4 = 1	30	4
O;B;1	O	0	350 - 1*30 = 320	B	1 - 1 = 0	30	1
S;A;10	S	10000	10	A			10
	S	10000	10	A	20	20	10
	S	10000 - 20*20 = 9600	10	A	20 - 10 = 10	20	10
S;A;10	S	9600 - 10*20 = 9400	10	A	10 - 10 = 0	20	10
S;J;1	S	9400	10	J	5	40	1
	S	9400 - 5*40 = 9200	10	J	5 - 1 = 4	40	1
S;J;4	S	9200 - 4*40 = 9040	10	J	4 - 4 = 0	40	4

En la variable fauces_destruida pasada por referencia se guardará un true si se pudieron destruir todas las vulnerabilidades, o false si no se pudo.
Se debe tener en cuenta que el archivo maestro de vulnerabilidades se puede abrir una única vez pero el maestro de escuadrones se puede abrir tantas veces como sea necesario.

Los caracteres que vamos a utilizar para representar los elementos de los ataques y debilidades de las vulnerabilidades serán los símbolos químicos de dichos elementos:

- 'H' para hidrógeno.
- 'I' para yodo.
- 'C' para carbono.
- 'N' para nitrógeno.
- 'O' para oxígeno.
- 'S' para azufre.
- 'P' para fósforo.

6. Especificaciones de corrección

Cuando falte 1 semana para la fecha de entrega se les dará el archivo pruebas.c que utiliza Kwyjibo para corregir los trabajos. Hasta ese momento deberán escribir sus propios pruebas.c para realizar las correspondientes

pruebas. Se debe poder compilar localmente la siguiente línea de comando sin errores:

gcc fauces.c pruebas.c -o pruebas -std=c99 -Wall -Wconversion -Werror

Se deberán aplicar todas las buenas prácticas de programación vistas hasta el momento, que como ya vimos son tan o más importantes que el correcto funcionamiento del código.
Importante: Los trabajos que no compilen **NO serán corregidos** e irán directo a reentrega. En caso de no realizar la entrega del trabajo también se irá a reentrega directo.

7. Entrega

Se debe subir un zip (sin importar el nombre) que contenga **exclusivamente** los archivos *fauces.c* y *fauces.h* a *Kwyjibo*. Se pueden realizar tantas entregas como se desee pero será tenida en cuenta la última y con límite de subida el día miércoles 22/11 a las 23:59 hs.
En caso de ir a reentrega, el límite será el día miércoles 06/12 a las 23:59 hs.
En ambos casos, la nota y devolución de los trabajos será enviada por mail.