TRABAJO OBLIGATORIO DE ESTRUCTURAS DE DATOS Y ALGORITMOS CURSO 2025

MANEJADOR DE VERSIONES

Consideraciones generales del obligatorio:

- **En el moodle se deja un archivo .zip que tiene ya armada la estructura del obligatorio, NO se pueden modificar los archivos .h a no ser por expresa confirmación del profesor en el que todos los estudiantes tendrán que modificarlos, dicha modificación se aclarará en clase y por el foro de moodle.**
- **El obligatorio debe ejecutar en el compilador cygwing sin excepción [1]**
- **Para compilar y y generar el ejecutable del obligatorio se utilizará el comando make y luego se ejecutará el archivo ejecutable principal.exe (windows), .\principal (linux) en las respectivas terminales. Los alumnos que entreguen tareas que no compilan en el mencionado compilador perderán el curso.**
- **Si se detecta copia en los obligatorios, todos los alumnos que tengan códigos "similares" perderán el curso.**
- **El obligatorio se debe realizar de forma individual, y no se permite utilizar nada que no se de en el curso, como por ejemplo el puntero nullptr.**
- **Para el primer obligatorio solo se consideran las funciones para las versiones de documentos principales, es decir versiones: 1, 2, 3, 4, etc. La segunda parte del obligatorio se debe considerar en su totalidad.**
- **El primer obligatorio se deberá entregar el día domingo 28 de septiembre antes de las 23:59 horas vía moodle y la defensa comenzará el lunes 29 de septiembre, (tener presente que no hay prórroga para la entrega).**
- **El segundo obligatorio se entregará el día domingo 2 de noviembre antes de las 23:59 horas vía moodle también y la defensa comenzará el día lunes 3 de noviembre.**
- **Todos los módulos deberán consumir la memoria exacta que se necesita, es decir no se podrá dejar memoria asignada sin usar, como por ejemplo, definir arreglos con tope y dejar parte del arreglo sin usar.**

Descripción del obligatorio:

El obligatorio consiste en la construcción de un sistema para modelar un manejador de versiones para archivos de texto.

CARACTERÍSTICAS GENERALES:

1) El manejador sólo permitirá trabajar con un archivo de texto y sus diferentes versiones.
2) Las versiones se identifican de manera unívoca por una secuencia de 1 o más números separados por un punto. Estos números pueden ser de más de 1 dígito y no hay límite en la secuencia de números. Las versiones nunca empiezan ni terminan con un punto. Algunos ejemplos son: 4, 2.3, 4.12.3.21, 1.2.3.4.5.6.7.8.9.10.11
3) Cada versión puede tener 0, 1 ó más subversiones dependientes.
4) Una versión depende y tiene como padre una sola versión, excepto las versiones iniciales del archivo (correspondientes a los números: 1, 2, 3…).
5) Cada versión X de un archivo incorpora 0, 1 o más modificaciones a la versión padre (si existe) de X.

6) En el sistema se distingue el uso de minúsculas y mayúsculas. Por ejemplo CASA ≠ Casa ≠ casa.

7) Los comandos se ejecutarán tal cual se muestran en los ejemplos con la misma cantidad de parámetro y con el tipo de dato y no se ingresa ningún comando fuera de ese formato, por lo que no hay que controlarlo. En caso de ejecutar un comando distinto a los mostrados simplemente saltara una línea sin hacer nada cosa que el código entregado ya lo realiza.

Ejemplo 1:

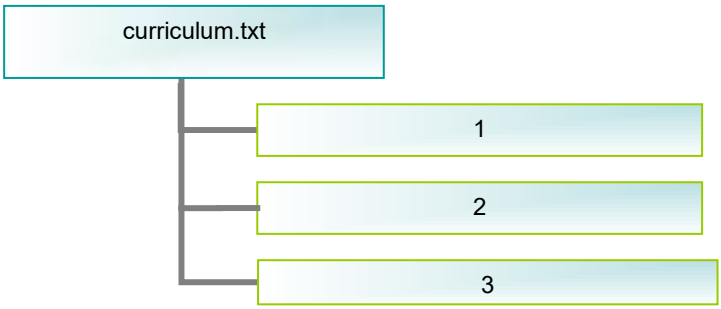

Diagrama Obligatorio 1

Ejemplo 2:

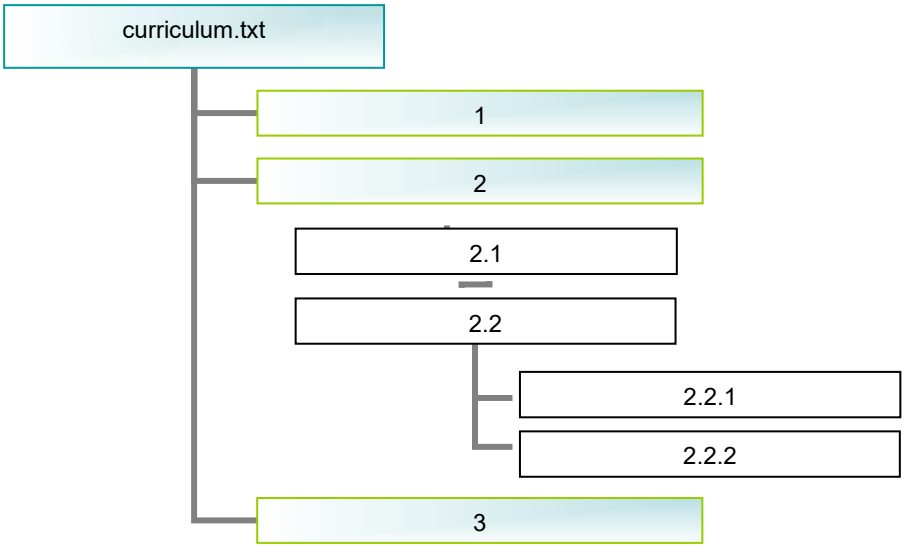

Diagrama Obligatorio 2

En el ejemplo 2, el esquema representa una posible estructura de versiones de un archivo con nombre "curriculum.txt". Este archivo posee tres versiones (1, 2 y 3). A su vez, la versión 2 contiene 2 subversiones (2.1 y 2.2) y finalmente, la 2.2 tiene como subversiones a 2.2.1 y 2.2.2.

Tipos de datos a manejar:

TipoRet	`enum _retorno{`
	`  OK, ERROR, NO_IMPLEMENTADA`
	`};`
	
	`typedef enum _retorno TipoRet;`

Archivo	`struct _rep_archivo{`
	`  /* aquí deben figurar los campos que usted considere necesarios para manipular el archivo con sus versiones */`
	`};`
	
	`en el .h de archivo se deberá realizar la siguiente definición de alias:`
	`typedef __rep_archivo* Archivo;`

Pueden definirse tipos de datos (estructuras de datos) auxiliares.

El sistema debe permitir realizar las operaciones que se detallan a continuación. Cada operación, excepto `CrearArchivo`, puede ser ejecutada exitosamente, retornando `OK`, ó puede generar un error, retornando `ERROR`, e imprimiendo: *ERROR: mensaje*. Los mensajes posibles para cada operación del sistema serán indicados. En caso de producirse un error, la estructura que representa a las versiones de un archivo (de tipo `Archivo`) permanecerá inalterada.
En la descripción de cada operación se incluye al menos un ejemplo. ***En todas las operaciones, excepto en la primera, se retoma el resultado del ejemplo del caso anterior.***

OPERACIONES RELATIVAS A LA CREACIÓN Y DESTRUCCIÓN DE UN ARCHIVO:

1) Crear el archivo.

   `Archivo CrearArchivo(char * nombre);` **(Obligatorio 1 y 2)**

   Crea el archivo con el nombre especificado y lo inicializa sin contenido (vacío). El archivo creado es retornado.
   Esta operación se ejecuta al inicio de una sesión de trabajo con un archivo.

   Ejemplo:

   ```
   Archivo a = CrearArchivo("currículo.txt")

   > mostrarVersiones(a)

Salida:

   curriculum.txt

   No hay versiones creadas
   ```

   Nota: `MostrarVersiones` se explica más adelante.

2) Borrar el archivo.

   `TipoRet BorrarArchivo(Archivo &a);` **(Obligatorio 1 y 2)**

   Elimina toda la memoria utilizada por el archivo y asigna NULL al puntero `a`. Se asume como precondición que **`a`** referencia a un archivo (en particular **`a`** es distinto a NULL).

Esta operación se ejecuta al final de una sesión de trabajo con un archivo.

Retornos posibles:	
OK	Siempre retorna OK.
ERROR	No existen errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

OPERACIONES RELATIVAS A LAS VERSIONES:

En las siguientes operaciones asumimos como precondición general que el archivo parámetro (de tipo puntero) está definido, en particular tiene nombre asignado (aunque su contenido sea eventualmente vacío y no tenga versiones).

3) Crear una nueva versión.

```
TipoRet CrearVersion(Archivo &a, char * version); (obligatorio 1 y 2)
```

Crea una nueva versión del archivo si la versión especificada cumple con las siguientes reglas:

- El padre de la nueva versión a crear ya debe existir. Por ejemplo, si creo la versión 2.15.1, la versión 2.15 ya debe existir.
  Las versiones del primer nivel no siguen esta regla, ya que no tienen versión padre.
- No pueden quedar "huecos" entre versiones hermanas. Por ejemplo, si creamos la versión 2.15.3, las versiones 2.15.1 y 2.15.2 ya deben existir.

Si la versión especificada ya existe, entonces se inserta haciendo que los hermanos (las versiones hermanas) se corran una posición a la derecha (aumentando una unidad). Por ejemplo si existen las versiones 2.15.1, 2.15.2 y 2.15.3, y creo la 2.15.2, las versiones 2.15.2 y 2.15.3 existentes pasan a ser 2.15.3 y 2.15.4 respectivamente, para poder incorporar la versión 2.15.2 nueva (este corrimiento influye en la numeración de todos los hijos –las subversiones– de las versiones afectadas).
Cada subversión de una versión X de un archivo incorpora, eventualmente, cambios (inserción y/o supresión de líneas) a la versión X. Ver las operaciones relativas al texto que se describen más adelante en este documento.

Ejemplo:

```
> crearVersion(a,1)
> crearVersion(a,2)
> crearVersion(a,2.1)
> mostrarVersiones(a)
```

Salida:

```
curriculum.txt

1
2
      2.1
```

Retornos posibles:	
OK	Si se pudo crear la nueva versión con éxito.
ERROR	Si la versión padre no existe (ver arriba). Si las subversiones anteriores del mismo padre no existen (ver arriba).
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

4) Borrar una versión, junto con sus hijos (subversiones), liberando toda la memoria involucrada.

`TipoRet BorrarVersion(Archivo &a, char * version); (Obligatorio 2)`

Elimina una versión del archivo si la `version` pasada por parámetro existe. En otro caso la operación quedará sin efecto. Si la versión a eliminar posee subversiones, éstas deberán ser eliminadas también, así como el texto asociado a cada una de las versiones. El texto será explicado más adelante.

No deben quedar números de versiones libres sin usar. Por lo tanto cuando se elimina una versión, las versiones hermanas que le siguen deben decrementar su numeración (así también sus subversiones dependientes). Por ejemplo, si existen las versiones 2.15.1, 2.15.2 y 2.15.3, y elimino la 2.15.1, la versión 2.15.2 y la 2.15.3 pasan a ser 2.15.1 y 2.15.2 respectivamente, esto incluye a todas las subversiones de estas versiones.

Si el archivo posee únicamente una versión (la 1), al eliminarla el archivo quedará vacío como cuando fue creado, es decir, únicamente con su nombre y contenido nulo.

Ejemplo:

```
> BorrarVersion(a,1)
> MostrarVersiones(a)
```

Salida:

```
curriculum.txt

1
        1.1
```

Retornos posibles:	
`OK`	Si se pudo eliminar la versión con éxito.
`ERROR`	Si `version` no existe.
`NO_IMPLEMENTADA`	Cuando aún no se implementó. Es el tipo de retorno por defecto.

5) Mostrar todas las versiones de un archivo de forma jerárquica.

`TipoRet MostrarVersiones(Archivo a);`

(obligatorio 1 y 2)

FORMATO: En primer lugar muestra el nombre del archivo. Después de una línea en blanco lista todos las versiones del archivo ordenadas por nivel jerárquico e indentadas según muestra el siguiente ejemplo (cada nivel está indentado por un tabulador).

Ejemplo:

```
> CrearVersion(a,1.2)
> CrearVersion(a,2)
> CrearVersion(a,1.2.1)
> MostrarVersiones(a)
```

Salida:

```
curriculum.txt

1
        1.1
        1.2
                1.2.1
2
```

Si el archivo no contiene versiones se mostrará la siguiente salida.

```
Salida:

    curriculum.txt

    No hay versiones creadas
```

Retornos posibles:	
OK	Siempre retorna OK.
ERROR	No existen errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

OPERACIONES RELATIVAS AL TEXTO:

6) Insertar una línea de texto, en cierta posición, a una versión de un archivo.

```
TipoRet InsertarLinea(Archivo &a, char * version, char * linea, unsigned int nroLinea);
```

(obligatorio 1 y 2)

Esta función inserta una `linea` de texto a la `version` parámetro en la posición `nroLinea`.
El número de línea debe estar entre 1 y n+1, siendo n la cantidad de líneas del archivo. Por ejemplo, si el texto tiene 7 líneas, se podrá insertar en las posiciones 1 (al comienzo) a 8 (al final).
Si se inserta en un número de línea existente, ésta y las siguientes líneas se correrán hacia adelante (abajo) dejando el espacio para la nueva línea.
No se puede insertar una línea en una versión que tenga subversiones.
Notar que el crear un archivo, éste no es editable hasta que no se crea al menos una versión del mismo. Sólo las versiones de un archivo son editables (se pueden insertar o suprimir líneas), siempre que no tengan subversiones creadas.

Ejemplo:

```
> borrarVersion(1)
> borrarVersion(2)  // el archivo queda vacío y sin versiones
> crearVersion(1)
> crearVersion(2)
> insertarLinea(a,1,Dirección:_Rivera_1234,1)
> insertarLinea(a,1,Teléfono:_6111111,2)
> insertarLinea(a,1,Nombre:_Juan_Pérez,1)
> crearVersion(a,1.1);
> insertarLinea(a,1.1,Estado_Civil:_Soltero,4)
> mostrarTexto(a,1)
```

```
Salida:

    curriculum.txt - 1

    1     Nombre: Juan Pérez
    2     Dirección: Rivera 1234
    3     Teléfono: 6111111
```

```
> mostrarTexto(a,1.1)
```

Salida:

```
curriculum.txt - 1.1

1     Nombre: Juan Pérez
2     Dirección: Rivera 1234
3     Teléfono: 6111111
4     Estado Civil: Soltero
```

Retornos posibles:	
OK	Si se pudo insertar la línea en la posición especificada a la versión parámetro del archivo.
ERROR	Si `version` no existe. Si `nroLinea` no es válido. Si `version` tiene subversiones.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

7) Borrar una línea de texto, en cierta posición, a una versión de un archivo.

```
TipoRet BorrarLinea(Archivo &a, char *version, unsigned int nroLinea);
```

(Obligatorio 2)

Esta función elimina una línea de texto de la `version` del archivo `a` en la posición `nroLinea`.
El número de línea debe estar entre 1 y n, siendo n la cantidad de líneas del archivo. Por ejemplo, si el texto tiene 7 líneas, se podrán eliminar líneas de las posiciones 1 a 7.
Cuando se elimina una línea, las siguientes líneas se corren, decrementando en una unidad sus posiciones para ocupar el lugar de la línea borrada.
No se puede borrar una línea de una versión que tenga subversiones creadas.

Ejemplo:

```
borrarLinea(a,1.1,3)
borrarLinea(a,1.1,3)
mostrarTexto(a,1)
```

Salida:

```
curriculum.txt - 1

1     Nombre: Juan Pérez
2     Dirección: Rivera 1234
3     Teléfono: 6111111

> mostrarTexto(a,1.1)
```

Salida:

```
curriculum.txt - 1.1

1     Nombre: Juan Pérez
2     Dirección: Rivera 1234
```

Retornos posibles:	
OK	Si se pudo eliminar la línea con éxito
ERROR	Si `version` no existe. Si `nroLinea` no es válido. Si `version` tiene subversiones.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

8) Mostrar el texto completo de una versión.

`TipoRet MostrarTexto(Archivo a, char * version);` **(Obligatorio 1 y 2)**

Esta función muestra el texto completo de la `version`, teniendo en cuenta los cambios realizados en dicha versión y en las versiones ancestras, de la cual ella depende.

FORMATO: En primer lugar se muestra el nombre del archivo. Luego, separado por un guión se mostrará la `version`. Después de una línea en blanco lista todas las líneas del texto. Cada línea comienza con el número de línea y separado por un tabulador se mostrará el texto.

Ver ejemplos previos (operación 7, por ejemplo).

Si la versión no contiene líneas se mostrará la siguiente salida.

Ejemplo:

```
> mostrarTexto(a,2)
```

Salida:

```
curriculum.txt - 2

No contiene líneas
```

Retornos posibles:	
OK	Si se pudo mostrar el texto, aún cuando éste no contenga líneas.
ERROR	Si `version` no existe.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

9) Mostrar todas las modificaciones en el texto de una versión X, respecto a la versión padre de X.

`TipoRet MostrarCambios(Archivo a, char * version);` **(Obligatorio 2)**

Esta función muestra los cambios que se realizaron en el texto de la `version` parámetro, sin incluir los cambios realizados en las versiones ancestras de la cual dicha versión depende.

FORMATO: En primer lugar muestra el nombre del archivo. Luego, separado por un guión se mostrará la `version`. Después de una línea en blanco lista todos los cambios realizados al texto. Cada cambio se muestra en una nueva línea. Si es una inserción comenzará con **IL** y si es un borrado con **BL**. Luego separado por un tabulador se mostrará el número de la línea modificada y, sólo en el caso de una inserción, luego de otro tabulador el texto insertado.

Ejemplo:

```
> mostrarCambios(a,1)
```

Salida:

```
curriculum.txt - 1

IL    1     Dirección: Rivera 1234
IL    2     Teléfono: 6111111
IL    1     Nombre: Juan Pérez

> mostrarCambios(a,1.1)
```

Salida:

```
curriculum.txt - 1.1

IL    4     Estado Civil: Soltero
BL    3
BL    3
```

Si la versión no contiene modificaciones se mostrará la salida indicada en el siguiente ejemplo.

Ejemplo:

```
> mostrarCambios(a,2)
```

Salida:

```
curriculum.txt - 2

No se realizaron modificaciones
```

Retornos posibles:	
OK	Si se pudieron mostrar los cambios, aún cuando éstos sean inexistentes.
ERROR	Si `version` no existe.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

10) Chequear si dos versiones son iguales.

`TipoRet Iguales(Archivo a,char* version1, char * version2, bool &iguales);`

(Obligatorio 2)

Esta función asigna al parámetro booleano (`iguales`) el valor `true` si ambas versiones (`version1` y `version2`) del archivo tienen exactamente el mismo texto, y `false` en caso contrario.

Ejemplo:

```
> iguales(a,1,1.1,iguales)

// iguales tiene el valor false

> insertarLinea(a,1.1,Teléfono:_6111111,3)
> iguales(a,1,1.1,iguales)

// iguales tiene el valor true
```

Retornos posibles:	
OK	Si se pudieron comparar las versiones.
ERROR	Si `version1` o `version2` no existen.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

OPERACIÓN OPCIONAL:

11) Crea una versión independiente de otras existentes.

`TipoRet VersionIndependiente(Archivo &a, char * version);` **(Obligatorio 2)**

Esta función crea una nueva versión al final del primer nivel con todos los cambios de la `version` especificada y de sus versiones ancestras. La versión que se crea debe ser independiente de cualquier otra versión.
Por ejemplo, si creamos una versión independiente a partir de la 2.11.3, entonces se crea una nueva versión al final del primer nivel (si existen las versiones 1, 2, 3 y 4, entonces se crea la 5) con los cambios realizados a las versiones 2, 2.11 y 2.11.3.

Ejemplo:

```
mostrarVersiones(a)
```

Salida:

```
curriculum.txt

1
      1.1
2

crearVersion(a,1.1.1)
insertarLinea(a,1.1.1,CP:_11100,3)
versionIndependiente(a,1.1.1)
mostrarVersiones(a)
```

Salida:

```
curriculum.txt

1
      1.1
            1.1.1
2
3

MostrarCambios(a,3)
```

Salida:

```
curriculum.txt – 3

IL    1     Dirección: Rivera 1234
IL    2     Teléfono: 6111111
IL    1     Nombre: Juan Pérez
IL    4     Estado Civil: Soltero
BL    3
```

```
BL    3
IL    3     Teléfono: 6111111
IL    3     CP: 11100

MostrarTexto(a,3)
```

Salida:

```
curriculum.txt - 3

1     Nombre: Juan Pérez
2     Dirección: Rivera 1234
3     CP: 11100
4     Teléfono: 6111111
```

Retornos posibles:	
OK	Si se pudo crear la versión independiente.
ERROR	Si `version` no existe.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

CATEGORÍA DE OPERACIONES

TIPO 1	Operaciones **imprescindibles** para que el trabajo obligatorio sea corregido.
TIPO 2	Operaciones importantes. Estas serán probadas independientemente, siempre que estén correctamente implementadas las operaciones de TIPO 1.
OPCIONAL	Operación, que realizada correctamente, acumula puntos adicionales.

TIPO 1	TIPO 2	OPCIONAL
`1) CrearArchivo`	`2) BorrarArchivo`	`11) VersionIndependiente`
`3) CrearVersion`	`4) BorrarVersion`	
`5) MostrarVersiones`	`8) MostrarTexto`	
`6) InsertarLinea`	`10) Iguales`	
`7) BorrarLinea`		
`9) MostrarCambios`		

Referencias:

[1] https://www.cygwin.com/