TRABAJO OBLIGATORIO DE
ESTRUCTURAS DE DATOS Y ALGORITMOS
MINI FILE SYSTEM

Se desea construir un *simulador* del administrador de archivos y directorios – file system – de un sistema operativo, el cual debe implementar un conjunto de comandos básicos de manejo de archivos y directorios, con sintaxis y comportamientos similares al Sistema Operativo MS-DOS.

Administración de Directorios y Archivos

La estructura de directorios (conocidos como *carpetas* en la jerga de Windows) deberá contar con un *directorio raíz* (carpeta base), a partir del cual se podrán crear nuevos directorios y archivos. A su vez, estos nuevos directorios podrán contener también nuevos archivos y directorios, permitiendo múltiples niveles en la estructura de directorios. Haremos referencia al directorio RAIZ mediante la palabra reservada RAIZ (que será en realidad el string "RAIZ"), lo cual implica que no podrá existir otro directorio con ese nombre. Introducimos también la noción de *directorio actual*, la cual es fundamental para la ejecución de muchos de los comandos que describiremos más adelante.

Los archivos que administrará esta estructura serán de texto. Un archivo es identificado por un *nombre*, cuyo largo no puede exceder 15 caracteres, y una *extensión,* de entre 1 y 3 caracteres como máximo. Los caracteres válidos tanto para el nombre como para la extensión serán de tipo alfanumérico {a,b,c,...,z | 0...9} (diferenciando mayúsculas de minúsculas). Se utilizará un punto para separar el nombre de la extensión. Cabe destacar que los directorios no podrán contar con extensión, solamente serán identificados con un *nombre*, cuyo largo no podrá exceder de 15 caracteres alfanuméricos. Deberá existir además un atributo que identifique si se trata de un archivo de lectura/escritura ó de sólo lectura. Los directorios no poseen este atributo.

Ejemplo:

```
RAIZ
├──archivo111.txt
├──archivo112.txt
└──archivo113.txt
```

- El directorio RAIZ cuenta con tres archivos de extensión .txt

Tipos de datos a manejar:

Cadena	`typedef char* Cadena;`

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

Sistema	`struct _sistema{`
	`  /* aquí deben figurar los campos que usted`
	`  considere necesarios para manipular el sistema */`
	`};`
	
	`typedef _sistema* Sistema;`

Pueden, y deberán, definirse tipos de datos (estructuras de datos) auxiliares. Por ejemplo, para representar todo lo relativo a los archivos.

Toda operación del sistema debe retornar un elemento de tipo **`TipoRet`**. Si la operación se realizó exitosamente, deberá retornar `OK`; si la operación no se pudo realizar de forma exitosa deberá retornar `ERROR` e imprimir ***un mensaje de error correspondiente al error producido***; y finalmente, si la operación no fue implementada, deberá retornar `NO_IMPLEMENTADA`. En cualquier caso que la ejecución de una operación no sea satisfactoria (retorne ERROR), el estado del sistema (el file system) deberá permanecer inalterado.

El sistema debe permitir realizar las siguientes operaciones.

OPERACIONES RELATIVAS A DIRECTORIOS:

Comando DIR

Este comando *muestra* el contenido del directorio actual, que para esta parte del laboratorio será únicamente el directorio RAIZ. Más adelante se explicará lo relativo a archivos de lectura y escritura.

Ejemplo:

DIR

```
RAIZ
archivo111.txt      Lectura
archivo112.txt      Lectura/Escritura
archivo113.txt      Lectura/Escritura
```

OPERACIONES RELATIVAS A LOS ARCHIVOS:

Comando CREATEFILE Archivo

Este comando *crea* un nuevo archivo en el directorio actual. Por ejemplo, suponiendo que contamos con la estructura anterior si ejecutamos los siguientes comandos el resultado debería ser:

CREATEFILE archivo2.txt

DIR

```
RAIZ
archivo111.txt      Lectura
archivo112.txt      Lectura/Escritura
archivo113.txt      Lectura/Escritura
archivo2.txt         Lectura/Escritura
```

Como convenciones establecemos que en su creación:

- Todos los archivos serán de Lectura/Escritura.
- El contenido de los archivos será vacío.
- No se permitirá crear un archivo con el nombre y la extensión de uno ya existente en el directorio actual.

```
TipoRet CREATEFILE (Sistema &s, Cadena nombreArchivo);
```

Retornos posibles:	
OK	Si el archivo pudo ser creado exitosamente.
ERROR	Si ya existe un archivo con ese nombre completo en el directorio actual.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando DELETE Archivo

Este comando *elimina* un archivo del directorio actual, siempre y cuando sea de escritura. Por ejemplo, si deseamos eliminar el archivo creado anteriormente:

DELETE archivo2.txt

DIR

```
RAIZ
archivo111.txt      Lectura
archivo112.txt      Lectura/Escritura
archivo113.txt      Lectura/Escritura
```

```
TipoRet DELETE (Sistema &s, Cadena nombreArchivo);
```

Retornos posibles:	
OK	Si se pudo eliminar el archivo exitosamente.
ERROR	Si no existe un archivo con ese nombre en el directorio actual. Si el archivo existe, pero es de sólo lectura.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando ATTRIB Archivo Parámetros (+W, –W)

Este comando *cambia* los atributos de un archivo perteneciente al directorio actual. Por ejemplo, retomando la estructura del último ejemplo, si deseamos que el archivo "archivo112.txt" sea de sólo lectura, entonces:

ATTRIB archivo112.txt -W

DIR

```
RAIZ
archivo111.txt      Lectura
archivo112.txt      Lectura
archivo113.txt      Lectura/Escritura
```

De la misma manera, podremos reestablecerlo a Lectura/Escritura ejecutando el comando inverso:

ATTRIB archivo112.txt +W

DIR

```
RAIZ
archivo111.txt      Lectura
archivo112.txt      Lectura/Escritura
archivo113.txt      Lectura/Escritura
```

```
TipoRet ATTRIB (Sistema &s, Cadena nombreArchivo, Cadena parametro);
```

Retornos posibles:	
OK	Si el comando se ejecutó exitosamente. Los únicos parámetros posibles son "+W" y "-W".
ERROR	Si no existe un archivo con ese nombre en el directorio actual.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

MANIPULACIÓN DEL CONTENIDO DE LOS ARCHIVOS

Se desea implementar algunos comandos básicos que administren el contenido de los archivos del sistema. Dichos comandos deberán aplicarse sobre un archivo que pertenezca al directorio actual. Recordamos que los archivos serán de texto, en donde un texto se define como una secuencia de caracteres de largo acotado, definido por la constante TEXTO_MAX. Asimismo, tener en cuenta que sólo podrán modificarse archivos que sean de Lectura/Escritura.

Los comandos a implementar son los siguientes:

Comando IF NombreArchivo Texto

Agrega un texto al comienzo del archivo NombreArchivo. En caso de excederse el largo total permitido, el texto resultante será truncado sobre el final para no sobrepasar el largo total TEXTO_MAX.

Continuando con el ejemplo anterior, se estableció el valor de la constante TEXTO_MAX en 22 y el archivo "archivo112.txt" es de Lectura/Escritura, el comportamiento esperado al ejecutar los siguientes comandos es (el comando TYPE, que muestra el contenido de un archivo, se describe más adelante):

IF archivo112.txt "Los peces del estanque son azules y naranjas."

TYPE archivo112.txt

```
Los peces del estanque
```

Luego, si ejecutamos los siguientes comandos, obtenemos:

IF archivo112.txt "Contemplemos "

TYPE archivo112.txt

```
Contemplemos Los peces
```

`TipoRet IF (Sistema &s, Cadena nombreArchivo, Cadena texto);`

Retornos posibles:	
OK	Si se pudo insertar el texto al comienzo del archivo, aún cuando esto implique truncar el contenido del archivo o incluso del texto parámetros.
ERROR	Si no existe un archivo con ese nombre en el directorio actual. Si el archivo es de sólo lectura.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando DF NombreArchivo K

Elimina los a lo sumo K primeros caracteres del archivo parámetro.

Sobre el ejemplo previo,

DF archivo112.txt 13

TYPE archivo112.txt

```
Los peces
```

DF archivo112.txt 50

TYPE archivo112.txt

```
El archivo no posee contenido
```

`TipoRet DF (Sistema &s, Cadena nombreArchivo, int k);`

Retornos posibles:	
OK	Si se pudieron eliminar los a lo sumo k primeros caracteres del archivo parámetro, aún cuando k fuera mayor o igual al largo del texto del archivo en cuestión (en cuyo caso el contenido quedaría vacío).
ERROR	Si no existe un archivo con ese nombre en el directorio actual. Si el archivo es de sólo lectura.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando TYPE NombreArchivo

Imprime el contenido del archivo parámetro, independientemente de su condición de Lectura/Escritura.

```
TipoRet TYPE (Sistema &s, Cadena nombreArchivo);
```

Retornos posibles:	
OK	Si se pudo mostrar el contenido del archivo (aún cuando éste fuere nulo).
ERROR	Si no existe un archivo con ese nombre en el directorio actual. No es un error el caso de un archivo con contenido vacío. En este caso deberá mostrar un mensaje que indique que el archivo parámetro no posee contenido.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

OPERACIONES GENERALES

Comando CREARSISTEMA

Inicializa el sistema para que contenga únicamente al directorio RAIZ, sin subdirectorios ni archivos. Este comando será usado al comienzo de un programa (set de pruebas) que simule una sesión del file system.

```
TipoRet CREARSISTEMA(Sistema &s);
```

Retornos posibles:	
OK	Si se pudo inicializar el sistema correctamente.
ERROR	Nunca retorna error.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando DESTRUIRSISTEMA

Destruye el sistema, liberando la memoria asignada a las estructuras que datos que constituyen el file system. Este comando será usado al final de un programa (set de pruebas) que simule una sesión del file system.

```
TipoRet DESTRUIRSISTEMA(Sistema &s);
```

Retornos posibles:	
OK	Si se pudo destruir el sistema correctamente.
ERROR	Nunca retorna error.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

COMANDOS EJECUTADOS ERRONEAMENTE

Aunque lo siguiente fue expresado al comienzo de este documento, decidimos sintetizar aquí la política de manejo de errores frente a la ejecución de las operaciones (los comandos) del sistema. Cada operación tiene determinadas precondiciones para que funcione correctamente. Las mismas se desprenden de la descripción dada de cada uno de los respectivos comandos. En caso de que alguna de estas precondiciones no se cumpla la operación debería retornar ERROR (tal cuál se señala para cada operación), imprimiendo además por pantalla un texto breve apropiado a cada caso para

destacar el tipo de error ocurrido. En cualquier caso que la ejecución de un comando no sea satisfactoria (retorne ERROR), el estado del sistema (el file system) deberá permanecer inalterado.

SOBRE LOS CHEQUEOS

Se permite considerar que la sintaxis de la entrada que recibirá el programa es válida. Esto quiere decir que no se requiere la realización de chequeos como los siguientes:

- Nombres de largo superior al especificado en la letra del obligatorio.
- Formatos inválidos para nombres de archivos, directorios o rutas.
- Parámetros inválidos para la operación ATTRIB.

Esto no quiere decir que no se deban chequear las condiciones establecidas para los comandos en la letra del obligatorio. Por ejemplo supresión de un archivo inexistente, etc.

Se pide:

- Implementar el código del sistema presentado, de acuerdo a las prácticas presentadas en el curso.
- Incluir las pre- y pos-condiciones de las operaciones.

La entrega se podrá realizar hasta el **Martes 11 de octubre a las 12.00hs**.

Ese mismo día en horario habitual de clase será la defensa.

La tarea se podrá realizar individualmente o en parejas, como se desee.