TRABAJO OBLIGATORIO 1 PRIMERA PARTE

ESTRUCTURAS DE DATOS Y ALGORITMOS
SISTEMA DE ARCHIVOS Y CARPETAS

Se desea construir un simulador del Sistema de Archivos y Carpetas de un Sistema Operativo, el cual debe implementar un conjunto de comandos básicos para su manejo. Este tendrá una sintaxis similar a la utilizada por la consola de Windows (DOS).

Administración de Directorios y Archivos

La estructura de directorios (conocidos como *carpetas* en la jerga de Windows) deberá contar con un directorio *RAIZ* (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. En nuestro simulador los archivos y la estructura de directorios se manejarán únicamente a nivel de memoria y no a nivel de disco.

Los archivos que administrará el sistema 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,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, serán identificados solamente con un *nombre*, cuyo largo no podrá exceder de 15 caracteres alfanuméricos, salvo el *directorio RAIZ* (nivel superior de la estructura de directorios) que se denota con el símbolo "/".

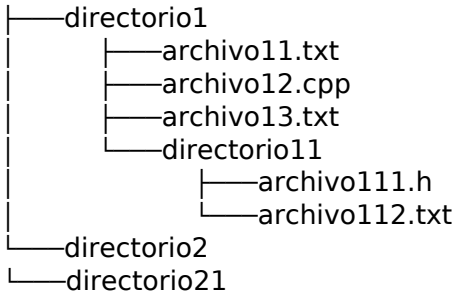
Los directorios pueden tener o no un tamaño máximo (cota) para los mismos. El tamaño de un archivo está determinado por la cantidad de caracteres que posee su contenido. El tamaño de un directorio en un momento dado está determinado por la suma de los tamaños de los archivos que se encuentran en dicho directorio (no en sus subdirectorios). Este tamaño nunca podrá exceder el tamaño máximo del directorio, si éste posee una cota.

Nomenclatura utilizada:

- Un *subdirectorio* de un *directorio* dado es un *directorio* que pertenece, en un nivel inferior, al *directorio* dado.
- Un *directorio* puede contener, tanto *subdirectorios* (generando una estructura arborescente) como *archivos*.
- En todo momento se está posicionado en un *directorio* específico dentro de la estructura, este es denominado *directorio actual*.

- Un *camino* es una secuencia de *directorios*, donde para cada pareja consecutiva de la secuencia existe una relación de padre a hijo, entre el primer *directorio* y el segundo.
- Una *ruta* está definida como un *camino* entre dos *directorios*. Esta puede servir para referirse tanto a *archivos* como a *directorios*.
- Una *ruta absoluta* es un *camino* desde la *RAIZ* hasta el *directorio* que se desee referir.
- Una *ruta relativa* es un *camino* desde el *directorio actual* hasta el *directorio* que se desee referir.

Ejemplo:



- "directorio1" es un directorio, contiene 3 archivos ("archivo11.txt", "archivo12.cpp" y "archivo13.txt") y 1 subdirectorio ("directorio11")
- a su vez "directorio1" y "directorio11" son subdirectorios del directorio RAIZ ("/")
- para referirnos al archivo "archivo111.h", si estamos situados en "directorio1" (o sea, éste es el directorio actual), podemos hacerlo mediante la ruta absoluta: "/directorio1/directorio11/archivo111.h" o relativa: "directorio11/archivo111.h"

Tipos de datos a manejar:

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 de directorios */
	};
	
	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.

Para la administración de la estructura de directorios y archivos se deberá implementar una serie de comandos, los cuales desarrollamos a continuación.

Intérprete de comandos

La *estructura de directorios/archivos* podrá ser modificada o consultada mediante la ejecución de una serie de *comandos*.

Notas:

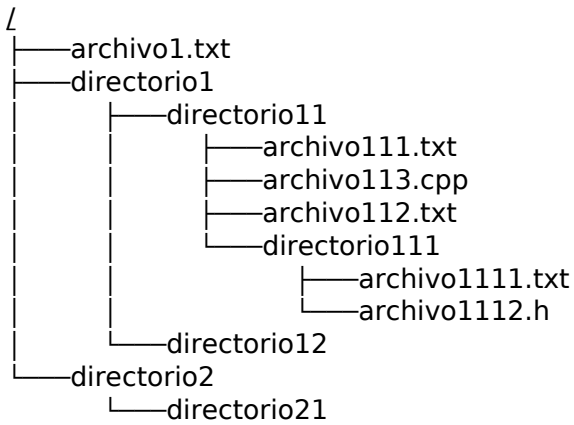
- Al comenzar, el *directorio actual* es el *directorio RAIZ*, que es el único componente de la estructura (no hay *archivos* ni otros *directorios*). El directorio *RAIZ* al inicio es no acotado (su tamaño máximo no está explícitamente restringido).
- Los nombres de los comandos están dados en mayúsculas.
- En todos los comandos que reciben parámetros que especifican *archivos* y/o *directorios*, éstos pueden ser determinados por *rutas relativas* o *absolutas*.

MANIPULACIÓN DE DIRECTORIOS

DIR

Este comando *muestra el contenido* del *directorio actual*, ya sean *subdirectorios* o *archivos*.
Nota: En esta primera parte sólo se manejará un directorio, el directorio raíz donde se crearán todos los archivos.
El comando acepta el parámetro **S** en la forma que veremos en la segunda parte.

Ejemplo: supongamos que contamos con la siguiente estructura y estamos posicionados en el *directorio RAIZ*.



La ejecución del comando sin parámetros, debería generar la siguiente salida por pantalla, mostrando el contenido del *directorio actual* (la *RAIZ* en este caso).

DIR ↵

```
archivo1.txt    Archivo    125
directorio1     Directorio  70
directorio2     Directorio
```

Asumimos que: 125 es el tamaño de del archivo archivo1.txt; 70 es el porcentaje ocupado del directorio "/directorio1"; y, el directorio "/directorio2" no posee cota máxima de tamaño.

Notas:

- El comando **DIR** realizará un listado de la información contenida en el *directorio actual*.
- Se debe distinguir entre *directorios* y *archivos*.
- En primer lugar serán listados los *archivos*, en orden alfabético, y luego los *directorios* también en orden alfabético. El orden será determinado sobre el string del nombre y la extensión del *archivo*.
- El formato de salida debe ser: 5 espacios entre el nombre del *directorio* o el *archivo* y la especificación [Directorio|Archivo]. Luego 5 espacios más y el tamaño del archivo (si es un archivo) ó el porcentaje ocupado del directorio (si es un directorio y tiene cota).

TipoRet DIR (Sistema &s , string parametro);

Retornos posibles:	
OK	● Siempre retorna OK.
ERROR	● No existe error posible. El parámetro debe ser o bien el string vacía o bien el string "/S" (barra S). *(Esto se explicará en la segunda parte del obligatorio)* No es un error si no existen archivos o subdirectorios en el directorio actual. Se deberá mostrar, en este caso, la ruta actual y un mensaje que indique que no hay archivos ni directorios.
NO_IMPLEMENTADA	● Cuando aún no se implementó. Es el tipo de retorno por defecto.

MANIPULACIÓN DE ARCHIVOS

Comando CREATE Archivo

Este comando *crea un nuevo archivo vacío* en el *directorio* especificado por su *ruta* (ya sea *relativa* o *absoluta*). Por ejemplo, suponiendo que el *directorio actual* es el *directorio RAIZ* y contamos con la estructura anterior:

CREATE /archivo2.txt ↵
DIR ↵

```
archivo1.txt    Archivo    125
archivo2.txt    Archivo    0
directorio1     Directorio  70
directorio2     Directorio
```

Asumimos que: 125 es el tamaño del archivo archivo1.txt; 70 es el porcentaje ocupado del directorio "/directorio1"; y, el directorio "/directorio2" no posee cota máxima de tamaño.
Cabe destacar que la ejecución del siguiente comando tendrá el mismo efecto sobre la estructura original:

CREATE archivo2.txt ↵

La estructura resultante en cualquiera de los dos casos es:

```
/
├──archivo2.txt
├──archivo1.txt
├──directorio1
│       ├──directorio12
│       └──directorio11
│              ├──archivo111.txt
│              ├──archivo112.txt
│              └──directorio111
│                     ├──archivo1111.txt
│                     └──archivo1112.h
└──directorio2
       ├──archivo113.cpp
       ├──directorio21
       └──directorio11
              ├──archivo113.cpp
              ├──archivo111.txt
              ├──archivo112.txt
              └──directorio111
                     ├──archivo1111.txt
                     └──archivo1112.h
```

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 de uno ya existente en el directorio actual.

TipoRet CREATE (Sistema &s, string 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 Sistema de Archivos. El *archivo* a ser borrado puede ser especificado mediante una *ruta relativa* o *absoluta.*

Ejemplo: si deseamos eliminar el *archivo* creado anteriormente:

DELETE archivo2.txt ↵
DIR ↵

```
archivo1.txt     Archivo     125
directorio1      Directorio   70
directorio2      Directorio
```

Asumimos que: 125 es el tamaño de del archivo archivo1.txt; 70 es el porcentaje ocupado del directorio "/directorio1"; y, el directorio "/directorio2" no posee cota máxima de tamaño.

TipoRet DELETE (Sistema &s, string nombreArchivo);

Retornos posibles:	
OK	● Si se pudo eliminar el archivo exitosamente.
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.

Comando UNDELETE

Este comando *recupera el último archivo eliminado* en el sistema mediante el comando **DELETE** (si hay al menos uno, sino no tiene efecto), restaurándolo al *directorio* en donde fue eliminado, siempre y cuando este *directorio*, determinado por su *ruta*, exista.

Ejemplo: según la ejecución del comando anterior:

DIR ↵

```
archivo1.txt     Archivo     125
directorio1      Directorio   70
directorio2      Directorio
```

Ahora, si ejecutamos este comando:

UNDELETE ↵

Recuperamos el *archivo* eliminado recientemente, "archivo2.txt"

El contenido del *directorio* sería el siguiente:

DIR ↵

```
archivo1.txt     Archivo     125
archivo2.txt     Archivo     0
directorio1      Directorio    70
directorio2      Directorio
```

Como convenciones establecemos que la ejecución de este comando podría no ser exitosa por los siguientes motivos:

- Que la estructura de directorios a la cual pertenecía el *archivo* haya sido alterada (se cambió la *ruta* desde el *directorio RAIZ* al *directorio* del cual se eliminó el *archivo* que se intenta recuperar).
- Que en el *directorio* a restaurar exista otro *archivo* con el mismo nombre y extensión.

De ocurrir cualquiera de las dos situaciones previas, se descarta la recuperación del *archivo* eliminándolo definitivamente del sistema de recuperación de archivos.

Nota: La cantidad de archivos que se podrán recuperar aplicando comandos **UNDELETE** no está explícitamente restringida.

TipoRet UNDELETE (Sistema &s);

Retornos posibles:	
OK	● Si se pudo recuperar el archivo exitosamente.
ERROR	● Si no existe el archivo a recuperar ya sea porque nunca se creó o porque se dan las condiciones establecidas previamente.
NO_IMPLEMENTADA	● Cuando aún no se implementó. Es el tipo de retorno por defecto.

MANIPULACIÓN DEL CONTENIDO DE LOS ARCHIVOS

Se desea implementar, además, una serie de comandos que administren el contenido de los *archivos* del Sistema. Recordemos que los *archivos* serán de texto, en donde un texto se define como una secuencia de líneas de largo acotado, definido por la constante LARGO_MAX.

Para dicha administración se desean implementar los siguientes comandos, que deben recibir como parámetro la *ruta relativa* o *absoluta* de un *archivo* existente en el sistema:

Comando IC NombreArchivo “Linea a insertar”

Inserta una línea al comienzo del texto del *archivo* NombreArchivo (la información a insertar debe ir entre comillas (“), el texto a su vez no podrá contener ese caracter).
Recordar que si un directorio está acotado, su tamaño máximo no puede ser excedido.

Continuando con el ejemplo anterior, si el directorio actual es “directorio11” y se estableció el valor de la constante TEXTO_MAX en 50, el comportamiento esperado al ejecutar los siguientes comandos es (el comando TYPE, que muestra el contenido de un archivo, se describe más adelante):

IC archivo112.txt “Los peces del estanque son azules y naranjas.”

TYPE archivo112.txt ↵

Los peces del estanque son azules y naranjas

Luego, si ejecutamos los siguientes comandos, obtenemos:
IC archivo112.txt “Contemplemos los peces del estanque son azules y naranjas.”
Error: El texto es mayor al soportado por el archivo.

Luego, si ejecutamos los siguientes comandos, obtenemos:

IC archivo112 “Veamos, ” ↵

TYPE archivo112 ↵

Veamos, Los peces del estanque son azules y naranjas

TipoRet IC (Sistema &s, string nombreArchivo, string texto);

Retornos posibles:	
OK	● Si se pudo insertar el texto al comienzo del archivo.
ERROR	● Si no existe un archivo con ese nombre en el directorio actual. ● Si la línea a insertar no está entre comillas.

	● Si no se pudo insertar el texto por superar el largo máximo establecido.
NO_IMPLEMENTADA	● Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando IF NombreArchivo “Linea a insertar”

Inserta una línea al final del texto del *archivo* NombreArchivo (vale la misma acotación que para el comando **IC**).
Recordar que si un directorio está acotado, su tamaño máximo no puede ser excedido.

TipoRet IF (Sistema &s, string nombreArchivo, string texto);

Retornos posibles:	
OK	● Si se pudo insertar el texto al final del archivo.
ERROR	● Si no existe un archivo con ese nombre en el directorio actual. ● Si la línea a insertar no está entre comillas. ● Si no se pudo insertar el texto por superar el largo máximo establecido.
NO_IMPLEMENTADA	● Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando BC NombreArchivo k

Borra, a lo sumo, las k primeras líneas de texto del *archivo* NombreArchivo. Si el texto tiene k o menos líneas, éste queda vacío.
Sobre el ejemplo previo,

BC archivo112.txt 1 ↵

TYPE archivo112.txt ↵

Los peces del estanque son azules y naranjas

TipoRet BC (Sistema &s, string nombreArchivo, int k);

Retornos posibles:	
OK	● Si se pudieron eliminar las a lo sumo k primeras líneas del archivo parámetro, aún cuando k fuera mayor o igual a la cantidad de líneas del archivo en cuestión (en cuyo caso el contenido quedaría vacío).
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.

Comando BF NombreArchivo k

Borra, a lo sumo, las k últimas líneas de texto del *archivo* NombreArchivo. Si el texto tiene k o menos líneas, éste queda vacío.

TipoRet BF (Sistema &s, string nombreArchivo, int k);

Retornos posibles:	
OK	● Si se pudieron eliminar las a lo sumo k últimas líneas del archivo parámetro, aún cuando k fuera mayor o igual a la cantidad de líneas del archivo en cuestión (en cuyo caso el contenido quedaría vacío).
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.

Comando CAT NombreArchivo1 NombreArchivo2

Concatena el *archivo* de nombre NombreArchivo2 a continuación del *archivo* de nombre NombreArchivo1, dejando el contenido del *archivo* resultante en el *archivo* NombreArchivo1. El *archivo* de nombre NombreArchivo2 no se modifica.
Recordar que si un directorio está acotado, su tamaño máximo no puede ser excedido.

TipoRet CAT (Sistema &s, string nombreArchivo1, string nombreArchivo2);

Retornos posibles:	
OK	● Si se pudieron concatenar los archivos.
ERROR	● Si no existe un archivo con el nombre NombreArchivo1 en el directorio actual. ● Si no existe un archivo con el nombre NombreArchivo2 en el directorio actual.
NO_IMPLEMENTADA	● Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando TYPE NombreArchivo

Muestra por pantalla el contenido del *archivo* NombreArchivo. Cada línea del *archivo* se desplegará en una línea nueva en la pantalla.

TipoRet TYPE (Sistema &s, string 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.

Comandos ejecutados erróneamente

Cada comando tiene determinadas precondiciones para que este 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 el resultado del comando será el siguiente:

El comando no se pudo ejecutar: “Precondición”.

Donde “Precondición” será un mensaje apropiado a cada caso.

Es también una precondición para cada una de los comandos del sistema que pueden recibir una *ruta* ya sea *relativa* o *absoluta*, que esta ruta completa exista en el sistema.

En cualquier caso que la ejecución de un comando no sea satisfactoria, el estado del sistema permanecerá 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.
- Utilización de caracteres no válidos (Ej: #, $, %) en el nombre de un archivo, directorio o ruta.
- La existencia de rutas como las siguientes: "//directorio" o "/directorio/".

Esto no quiere decir que no se deban chequear las condiciones establecidas para los comandos en la letra del obligatorio.

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	Operaciones, que realizadas correctamente serán tenidas en cuenta al final del curso siempre que las operaciones TIPO 1 y TIPO 2 están aprobadas.

TIPO 1	TIPO 2	OPCIONAL
DIR (sin parámetros)	**DELETE**	**IC**
CREATE	**BF**	**BC**
IF	**CAT**	**UNDELETE**
TYPE		

Fecha límite para la entrega de esta primera parte 10 de octubre del 2019 a las 23:59 horas.

La misma deberá ser enviada en un archivo comprimido que contenga todo el proyecto a la dirección de correo del docente (gualberto.hernandez@utec.edu.uy) donde en asunto debe indicar [LAB01-EDA] seguido del nombre y apellido de los integrantes del equipo.

El laboratorio se puede hacer en grupo de hasta 3 alumnos por equipo. Se debe comunicar al docente via correo electrónico a la dirección (gualberto.hernandez@utec.edu.uy), los integrantes del equipo (nombre y apellido) antes del 20 de setiembre, caso contrario, los alumnos que no formen parte de un equipo deberán realizarlo en forma individual.