TRABAJO OBLIGATORIO 2

ESTRUCTURAS DE DATOS Y ALGORITMOS
SISTEMA DE ARCHIVOS Y CARPETAS

Siguiendo con el trabajo propuesto para el Obligatorio 1, se desea agregar comandos al simulador del administrador de archivos y directorios – file system – de un sistema operativo. Los comandos que ya fueron implementados en el Obligatorio 1 continuarán disponibles, agregándose en esta entrega comandos para manipular la estructura de directorios.

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 "/".

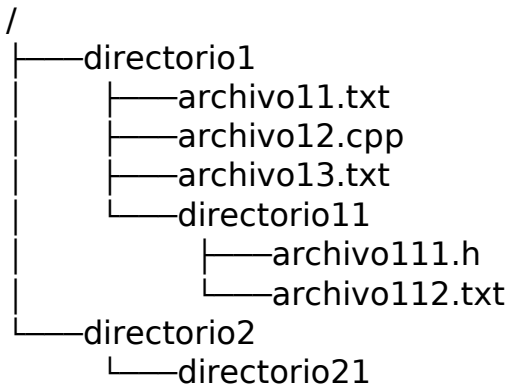
Los directorios pueden tener o no un tamaño máximo (cota) para los mismos. El tamaño de un archivo está determinado por la cantidad de caracteres que posee su contenido. El tamaño de un directorio en un momento dado está detrminado por la suma de los tamaños de los archivos que se encuentran en dicho 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"

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*.

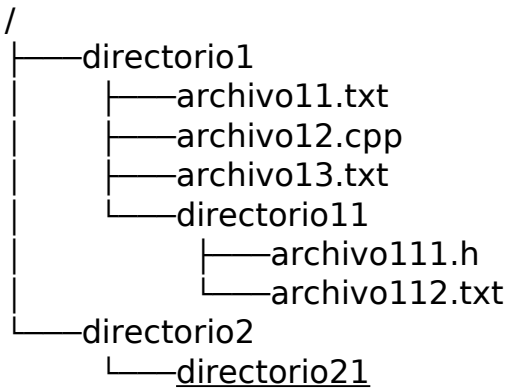
Notas:

- 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*.

Comando PWD

Este comando muestra el camino desde la *RAIZ* al *directorio actual* siguiendo el formato: "/.../.../dirActual".

Ejemplo: considerar la siguiente estructura de directorios y archivos, con "/directorio2/directorio21" como *directorio actual*.



La ejecución del comando **PWD** debe desplegar:

/directorio2/directorio21

TipoRet PWD (Sistema &s, Cadena nombreDirectorio);

Retornos posibles:	
OK	• Siempre retorna OK.
ERROR	• Nunca retorna error.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando CD Directorio

Este comando **es el único** que *permite desplazarnos en la estructura de directorios*, definiendo así al nuevo *directorio actual*.

Ejemplo: considerando la estructura del ejemplo anterior, si nos encontramos en el *directorio* "/directorio2/directorio21" (*directorio actual*) y queremos movernos al *directorio* "/directorio1":

CD /directorio1↵

De esta manera, el *directorio actual* pasará a ser "/directorio1". Este es un ejemplo de *ruta absoluta*, dado que el *directorio* destino está dado desde la *RAIZ*.

```
/
├──directorio1
│       ├──archivo11.txt
│       ├──archivo12.cpp
│       ├──archivo13.txt
│       └──directorio11
│              ├──archivo111.h
│              └──archivo112.txt
└──directorio2
        └──directorio21
```

Se debe además poder navegar en la estructura utilizando *rutas relativas*.

Ejemplo: Dado que el *directorio actual* es "/directorio1", podemos hacer que este sea "/directorio1/directorio11" ejecutando el siguiente comando:

CD directorio11 ↵

De esta manera, el *directorio actual* pasará a ser “/directorio1/directorio11”.

```
/
├──directorio1
│       ├──archivo11.txt
│       ├──archivo12.cpp
│       ├──archivo13.txt
│       └──directorio11
│              ├──archivo111.h
│              └──archivo112.txt
└──directorio2
        └──directorio21
```

Como convención establecemos que:

- No se podrá ejecutar el comando **CD** sobre un *directorio* inexistente en el Sistema.
- No se podrá ejecutar sobre un *archivo*.

TipoRet CD (Sistema &s, Cadena nombreDirectorio);

Retornos posibles:	
OK	• Si se pudo ejecutar exitosamente el comando CD.
ERROR	• Si no existe el subdirectorio destino. • Si se intenta ir al directorio padre y el directorio actual es el directorio RAIZ.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando MKDIR Directorio

Este comando *crea un nuevo directorio* en la *ruta* especificada. Esta *ruta* puede ser tanto *relativa* como *absoluta*.

Ejemplo: si el sistema se encuentra posicionado en "/directorio1/directorio11", es decir éste es el *directorio actual*, la ejecución del siguiente comando crea un nuevo *subdirectorio* de "/directorio1/directorio11":

MKDIR directorio111 ↵

Obteniendo como resultado la siguiente estructura:

```
/
├──directorio1
│      ├──archivo11.txt
│      ├──archivo12.cpp
│      ├──archivo13.txt
│      └──directorio11
│            ├──archivo111.h
│            ├──archivo112.txt
│            └──directorio111
└──directorio2
       └──directorio21
```

Ejemplo: También podemos crear un nuevo *directorio* mediante la utilización de una *ruta absoluta*. Si queremos crear el *directorio* "directorio112" como descendiente de "directorio11" podemos ejecutar el siguiente comando:

MKDIR /directorio1/directorio11/directorio112 ↵

Obteniendo como resultado la siguiente estructura:

```
/
├──directorio1
│      ├──archivo11.txt
│      ├──archivo12.cpp
│      ├──archivo13.txt
│      └──directorio11
│            ├──archivo111.h
│            ├──archivo112.txt
│            ├──directorio111
│            └──directorio112
└──directorio2
       └──directorio21
```

Como convenciones establecemos que:

- En un *directorio* no podrán existir dos *subdirectorio*s con el mismo nombre.
- El directorio creado es no acotado. Su tamaño máximo no está explícitamente restringido.

TipoRet MKDIR (Sistema &s, Cadena nombreDirectorio);

Retornos posibles:	
OK	• Si se pudo crear el directorio exitosamente.
ERROR	• Si el directorio actual ya contiene un subdirectorio con ese nombre. • Si nombreDirectorio es RAIZ.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando RMDIR Directorio

Este comando *elimina un directorio* especificado por una *ruta* (*relativa* o *absoluta*) además de todos los *subdirectorios* y *archivos* que éste contenga.
Ejemplo: Supongamos que tenemos la siguiente estructura de *directorios* en donde el *directorio actual* es "/directorio1".

```
/
├──directorio1
│      ├──archivo11.txt
│      ├──archivo12.cpp
│      ├──archivo13.txt
│      └──directorio11
│             ├──archivo111.h
│             ├──archivo112.txt
│             └──directorio111
└──directorio2
       └──directorio21
```

RMDIR directorio11 ↵

El resultado debería ser el siguiente:

```
/
├──directorio1
│      ├──archivo11.txt
│      ├──archivo12.cpp
│      └──archivo13.txt
└──directorio2
       └──directorio21
```

Como convención establecemos que:

- No se podrá aplicar este comando a un *directorio ancestro* del *directorio actual* (ni a él mismo).

  Ejemplo: Si se está posicionado en el *directorio* "/directorio2/directorio21", no se podrá ejecutar el siguiente comando:

RMDIR /directorio2 ↵

ni el siguiente:

RMDIR /directorio2/directorio21 ↵

- No se podrá aplicar este comando al *directorio RAIZ* (esto se debe a que nunca se puede estar posicionado en un nivel superior a la *RAIZ*).

TipoRet RMDIR (Sistema &s, Cadena nombreDirectorio);

Retornos posibles:	
OK	• Si se pudo eliminar el subdirectorio exitosamente.
ERROR	• Si no existe el subdirectorio a eliminar.
NO_IMPLEMENTADA	• Cuando aún no se implementó. Es el tipo de retorno por defecto.

Comando DIR [Parámetro]

Este comando *muestra el contenido* del *directorio actual*, ya sean *subdirectorios* o *archivos*.
El comando acepta el parámetro **S** en la forma que veremos más adelante.

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

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

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 exactamente igual al del ejemplo anterior: 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).

Parámetro:

- **S** – Muestra la estructura de directorios a partir del *directorio* actual, organizada de la siguiente manera: primero se listan los *archivos* del *directorio actual* y luego el contenido de cada uno de los *subdirectorios* siguiendo el mismo procedimiento (recursivamente). Tanto el listado de *archivos* como el de *directorios* debe ser realizado en orden alfabético. Antes de listar el contenido de un directorio, se debe imprimir la etiqueta:
  <contenido:NombreDirectorio>, a su vez, cuando se llega al final del contenido de un directorio se debe imprimir la etiqueta </contenido:NombreDirectorio>. A diferencia del caso anterior (el **DIR** sin parámetro), no se imprimen aquí los tamaños de los archivos ni los porcentajes ocupados de los directorios con cota.

Ejemplo: Para el caso anterior, suponiendo que el directorio actual es la RAIZ "/", la impresión quedaría de la siguiente manera:

DIR S ↵

```
archivo1.txt     Archivo
directorio1     Directorio
<contenido:directorio1>
directorio11     Directorio
<contenido:directorio11>
archivo111.txt     Archivo
archivo112.txt     Archivo
archivo113.cpp     Archivo
directorio111     Directorio
```

```
<contenido:directorio111>
archivo1111.txt     Archivo
archivo1112.h     Archivo
</contenido:directorio111>
</contenido:directorio11>
directorio12     Directorio
<contenido:directorio12>
</contenido:directorio12>
</contenido:directorio1>
directorio2     Directorio
<contenido:directorio2>
directorio21     Directorio
<contenido:directorio21>
</contenido:directorio21>
</contenido:directorio2>
```

TipoRet DIR (Sistema &s, Cadena parametro);

Retornos posibles:	
OK	• Siempre retorna OK.
ERROR	• No existe error posible. El parámetro debe ser o bien la cadena vacía o bien la cadena "/S" (barra S). 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.

Comando COPY Directorio o Archivo a Directorio-Destino

Este comando *copia un directorio o archivo* desde un *directorio* origen hacia un *directorio* destino. Tanto el *directorio* origen como el *directorio* destino pueden estar especificados por sus *rutas* ya sean *relativas* o *absolutas*. Como es de suponerse, al copiar un *directorio,* su estructura de *subdirectorio*s debería quedar inalterada.

Ejemplo: supongamos que contamos con la siguiente estructura y estamos posicionados en el *directorio* "/directorio2" y deseamos copiar su *subdirectorio* "/directorio2/directorio11" en el *directorio* "/directorio1".

```
/
├──archivo1.txt
├──directorio1
│      └──directorio12
└──directorio2
```

```
        ├──archivo113.mod
        ├──directorio21
        └──directorio11
            ├──archivo111.txt
            ├──archivo112.txt
            └──directorio111
                ├──archivo1111.txt
                └──archivo1112.def
```

La sintaxis del comando debería ser en el caso descripto la siguiente:

COPY directorio11 /directorio1↵

Y el resultado esperado:

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

Para el caso de la copia de archivos presentamos el siguiente ejemplo: si nos encontramos posicionados en el *directorio* "/directorio2" y deseamos copiar el *archivo* "archivo113.cpp" hacia el *directorio* "/directorio2/directorio11", la sintaxis del comando debería ser la siguiente:

COPY archivo113.cpp directorio11↵

Y el resultado esperado:

```
/
├──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:

- Si se intenta copiar un *directorio* (o archivo) y el *directorio* destino cuenta con un *directorio* (o archivo) con el mismo nombre, éste se sobrescribirá.
- Si el directorio destino está acotado, su tamaño máximo no puede ser excedido.
- Por motivos de simplicidad, en el caso de la copia de directorios no se podrá tomar como *directorio* destino un *directorio* descendiente del *directorio* origen, ni a este último.

TipoRet COPY(Sistema &s, Cadena directorio/archivoOrigen, Cadena directorioDestino);

Retornos posibles:	
OK	• Si se pudo correctamente COPY.
ERROR	• Si directorio/archivoOrigen no existe. • Si directorioDestino no existe. • Si ya existe un directorio/archivoOrigen en el directorioDestino.
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
MKDIR	**PWD**	**DIR S**
CD	**RMDIR**	**COPY**
DIR		

Fecha límite para la entrega de esta primera parte 17 de noviembre de 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 [LAB02-EDA] seguido del nombre y apellido de los integrantes del equipo.

Se deben mantener los equipos ya formados para la primera entrega.