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

- "directorio1" es un directorio que contiene 2 archivos ("archivo11.txt", "archivo12.cpp") y 1 subdirectorio ("directorio11")
- a su vez "directorio1" y "directorio11" son subdirectorios del directorio RAIZ, como así también "directorio2" y "directorio21"
- el directorio actual es en el ejemplo: "directorio1"

Tipos de datos a manejar:

| | |
|---|---|
| **Cadena** | `typedef char* Cadena;` |
| **TipoRet** | `enum _retorno{`<br>`  OK, ERROR, NO_IMPLEMENTADA`<br>`};`<br>`typedef enum _retorno TipoRet;` |
| **Sistema** | `struct _sistema{`<br>`  /* aquí deben figurar los campos que usted considere necesarios para manipular el sistema de directorios */`<br>`};`<br>`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 LOS DIRECTORIOS:

## *Comando CD Directorio*

Este será el "***único***" comando que nos permitirá desplazarnos en la estructura de directorios. Por ejemplo, considerando la estructura del ejemplo inicial, si nos encontramos en el directorio "directorio1" y queremos movernos al directorio "directorio11":

**CD directorio11**

De esta manera, el directorio actual pasará a ser "directorio11". Como convención establecemos que solamente se podrá mover a un directorio hijo del *directorio actual* (no se puede bajar varios niveles en la estructura con la ejecución de un único CD).
Ahora, si deseamos volver al directorio "directorio1" o, en general, al padre del actual (siempre que el actual no sea el directorio RAIZ):

**CD ..**

Se debe poder regresar al directorio RAIZ desde cualquier otro directorio, para ello utilizaremos la siguiente sintaxis:

**CD RAIZ**

```
TipoRet CD (Sistema &s, Cadena nombreDirectorio);
```

| Retornos posibles: | |
|---|---|
| `OK` | • Si se pudo ejecutar exitosamente el comando CD. |
| `ERROR` | • Si no existe el subdirectorio destino.<br>• 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 subdirectorio vacío del *directorio actual*. Por ejemplo, si el sistema se encuentra posicionado en el directorio "directorio1",

```
RAIZ
└──directorio1
```

**MKDIR directorio11**

```
RAIZ
└──directorio1
        └──directorio11
```

Como convención establecemos que en un directorio no podrán existir dos subdirectorios con el mismo nombre.

**`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.<br>• 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 subdirectorio hijo del directorio actual.
Por ejemplo, si el sistema se encuentra en la situación del ejemplo anterior, posicionado en el directorio "directorio1", y se quiere eliminar el subdirectorio "directorio11", el comando y el resultado de la ejecución de este deberían ser los siguientes:

**RMDIR directorio11**

```
RAIZ
└──directorio1
```

Como convenciones, establecemos que:

- Al eliminar un directorio se eliminarán todos sus subdirectorios junto con los archivos correspondientes, independientemente de sus condiciones de Lectura/Escritura.
- El directorio RAIZ no podrá ser eliminado.

**`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 MOVE Directorio ó Archivo a Mover Directorio_Destino*

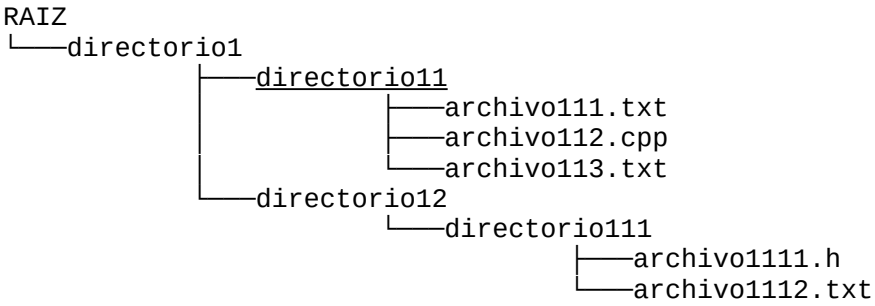
Este comando *mueve* un directorio o archivo desde su directorio origen hacia un nuevo directorio destino. El directorio origen debe ser un subdirectorio del directorio actual. Como es de suponerse, al mover un directorio, su estructura de subdirectorios debería quedar inalterada. Ejemplo: Supongamos la siguiente estructura.

```
RAIZ
└──directorio1
            ├──directorio11
            │           ├──archivo111.txt
            │           ├──archivo112.cpp
            │           ├──archivo113.txt
            │           └──directorio111
            │                       ├──archivo1111.h
            │                       └──archivo1112.txt
            │
            └──directorio12
```

Si nos encontramos posicionados en el directorio "directorio11" y deseamos mover su subdirectorio "directorio111" hacia el directorio "directorio12", la sintaxis del comando debería ser la siguiente:

**MOVE directorio111 RAIZ/directorio1/directorio12 ↵**

Y el resultado esperado:




Como convenciones establecemos que:

- Si se intenta mover un directorio (ó archivo) y el directorio destino cuenta con un directorio (ó archivo) con el mismo nombre, éste se sobrescribirá completamente.
- En el caso del MOVE para directorios, no se podrá tomar como directorio destino un subdirectorio del directorio origen, ni este último. Tampoco se permitirá la ejecución de este comando cuando se provocara la pérdida del directorio actual.
- El directorio destino *Directorio_Destino* deberá incluir el camino desde el directorio RAIZ hasta el directorio en cuestión en el formato: "RAIZ/.../.../DirXX", siendo DirXX el nombre del directorio destino.

**`TipoRet MOVE (Sistema &s, Cadena nombre, Cadena directorioDestino);`**

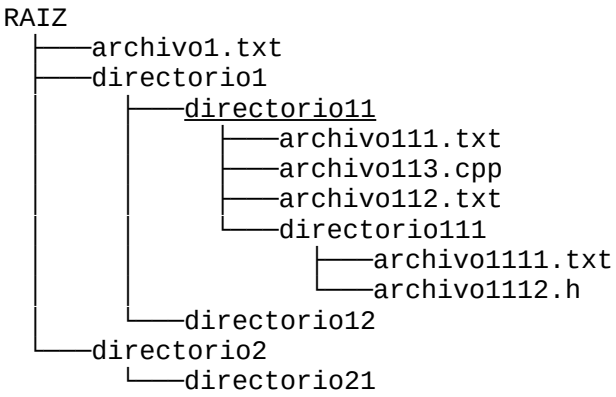
| Retornos posibles: | |
|---|---|
| `OK` | • Si se pudo mover correctamente. |
| `ERROR` | • Si no existe un directorio o archivo con ese nombre en el directorio actual.<br>• Si no existe el directorio destino.<br>• Si el directorio destino es un subdirectorio del directorio origen, o este último (en el caso del MOVE para directorios).<br>• Si se provoca la pérdida del directorio actual. |
| `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 "directorio11".



La ejecución del comando sin parámetros, debería generar la siguiente salida por pantalla, mostrando el contenido del directorio actual.

**DIR**

```
RAIZ/directorio1/directorio11
archivo111.txt     Lectura
archivo112.txt     Lectura/Escritura
archivo113.cpp     Lectura/Escritura
directorio111
```

Asumimos que el archivo "archivo111.txt" es de Lectura y los archivos "archivo112.txt" y "archivo113.cpp" son de Lectura/Escritura.

Como convenciones establecemos que:

- El comando DIR realizará un listado de la información contenida en el directorio actual.
- En primer lugar se imprimirá la ruta completa del directorio actual, desde el directorio RAIZ. Luego, serán listados los archivos, en orden alfabético creciente, y posteriormente los directorios también en orden alfabético. Para los archivos, el orden será determinado sobre el string del nombre y la extensión.
- El formato de salida debe ser exactamente igual al del ejemplo anterior. Si se trata de un archivo, luego del nombre completo irá el atributo "Lectura" o "Lectura/Escritura" (según el caso), separado por 5 espacios en blanco.

**Parámetro:**

**/S** – Muestra la estructura de directorios a partir del directorio actual, organizada de la siguiente manera: primero se imprime la ruta completa del directorio actual, desde el directorio RAIZ. Luego se listan los archivos del directorio actual y posteriormente 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 y debe incluirse en cada caso la ruta completa desde el directorio RAIZ. A diferencia del caso anterior (el DIR sin parámetro), no se imprimen aquí los atributos de los archivos.

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

**DIR /S**

```
RAIZ
RAIZ/archivo1.txt
RAIZ/directorio1
RAIZ/directorio1/directorio11
RAIZ/directorio1/directorio11/archivo111.txt
RAIZ/directorio1/directorio11/archivo112.txt
RAIZ/directorio1/directorio11/archivo113.cpp
RAIZ/directorio1/directorio11/directorio111
RAIZ/directorio1/directorio11/directorio111/archivo1111.txt
RAIZ/directorio1/directorio11/directorio111/archivo1112.h
RAIZ/directorio1/directorio12
RAIZ/directorio2
RAIZ/directorio2/directorio21
```

```
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. |

## 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 y el directorio actual es "directorio11", si ejecutamos los siguientes comandos el resultado debería ser:

**CREATEFILE archivo2.txt**

**DIR**

```
RAIZ/directorio1/directorio11
archivo111.txt    Lectura
archivo112.txt    Lectura/Escritura
archivo113.txt    Lectura/Escritura
archivo2.txt      Lectura/Escritura
directorio111
```

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 no sea de sólo lectura. Por ejemplo, si deseamos eliminar el archivo creado anteriormente:

**DELETE archivo2.txt**

**DIR**

```
RAIZ/directorio1/directorio11
archivo111.txt     Lectura
archivo112.txt     Lectura/Escritura
archivo113.txt     Lectura/Escritura
directorio111
```

```
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.<br>• 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 y considerando que el directorio actual es "directorio11", si deseamos que el archivo "archivo112.txt" sea de sólo lectura, entonces:

**ATTRIB archivo112.txt -W**

**DIR**

```
RAIZ/directorio1/directorio11
archivo111.txt     Lectura
archivo112.txt     Lectura
archivo113.txt     Lectura/Escritura
directorio111
```

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

**ATTRIB archivo112.txt +W**

**DIR**

```
RAIZ/directorio1/directorio11
archivo111.txt     Lectura
archivo112.txt     Lectura/Escritura
archivo113.txt     Lectura/Escritura
directorio111
```

```
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 desean implementar, además, 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 IC 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, si el directorio actual es “directorio11”, 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):

**IC archivo112.txt “Los peces”**

**TYPE archivo112.txt**

```
Los peces
```

## *Comando IF NombreArchivo Texto*

*Agrega* un texto al final 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, si el directorio actual es “directorio11”, 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 “del estanque”**

**TYPE archivo112.txt**

```
Los peces del estanque
```

Luego, si ejecutamos los siguientes comandos, obtenemos:

**IC 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.<br>• Si el archivo es de sólo lectura. |
| `NO_IMPLEMENTADA` | • Cuando aún no se implementó. Es el tipo de retorno por defecto. |

## *Comando DC NombreArchivo K*

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

Sobre el ejemplo previo,

**DC archivo112.txt 13**

**TYPE archivo112.txt**

```
Los peces
```

**DC archivo112.txt 50**

**TYPE archivo112.txt**

```

```

```
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.<br>• 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 últimos caracteres del archivo parámetro.

Sobre el ejemplo previo,

**DF archivo112.txt 6**

**TYPE archivo112.txt**

```
Los
```

**DF archivo112.txt 50**

**TYPE archivo112.txt**

```

```

```
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.<br>• 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.<br>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. |

## *Comando SEARCH NombreArchivo Texto*

*Busca dentro del archivo la existencia del texto.*

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

| Retornos posibles: | |
|---|---|
| OK | • Si se pudo encontrar el texto dentro del archivo. |
| ERROR | • Si no se pudo encontrar el texto dentro del archivo.<br>• 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 REPLACE NombreArchivo Texto1 Texto2*

*Busca y reemplaza dentro del archivo la existencia del texto1 por el texto2.*

```
TipoRet REPLACE (Sistema &s, Cadena nombreArchivo, Cadena texto1, Cadena texto2);
```

| Retornos posibles: | |
|---|---|
| OK | • Si se pudo reemplazar el texto dentro del archivo. |
| ERROR | • Si no se pudo encontrar o reemplazar el texto dentro del archivo.<br>• 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. |

# 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 comandos tales como CD, DIR o ATTRIB.

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

## 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, pueden acumular hasta 10 puntos adicionales para el curso. |

| TIPO 1 | TIPO 2 | OPCIONAL |
|---|---|---|
| CEARSISTEMA (1era. Entrega) | DIR "/S" (2da. Entrega) | |
| CD (2da. Entrega) | RMDIR (2da. Entrega) | MOVE |
| MKDIR (2da. Entrega) | DELETE (1era. Entrega) | REPLACE |
| DIR "" (1era. Entrega) | ATTRIB (1era. Entrega) | |
| CREATEFILE (1era. Entrega) | IC (1era. Entrega) | |
| IF (1era. Entrega) | DC (1era. Entrega) | |
| DF (1era. Entrega) | DESTRUIRSISTEMA (1era. Entrega) | |
| TYPE (1era. Entrega) | SEARCH (1era. Entrega) | |

## *A fin de construir las implementaciones anteriores se pide:*

1) Definición y desarrollo de las estructuras de datos involucradas, siguiendo la metodología basada en módulos (vista en el curso), y usando C con algunas extensiones de C++ como lenguaje de programación *(No se permite el uso de clases, goto, break, etc)*.
2) Implementación de los módulos. El código de todo el sistema debe estar desarrollado y comentado de acuerdo a las prácticas presentadas en el curso. Incluir las pre- y pos-condiciones de las operaciones.
3) Se debe entregar también un *makefile* que genere todo el proyecto.

## *Formas y Plazos de Entrega*

La tarea deberá realizarse en grupos de dos o tres estudiantes cada uno. Los grupos deben inscribirse llenando un formulario web hasta el 10 de octubre inclusive, los estudiantes que se anoten de forma individual se les asignará un grupo de forma aleatoria.

El plazo de ***entrega para la primera parte será hasta el domingo 29 de octubre inclusive.*** vía el Campus Virtual.
El plazo de ***entrega para la segunda parte será hasta el domingo 19 de noviembre inclusive*** vía el Campus Virtual.
En ambos casos deberán además coordinar una defensa para evaluar el trabajo realizado.
Quienes realicen las partes opcionales se les podrá asignar hasta 10 puntos extra en el puntaje del curso.