ESTRUCTURAS DE DATOS Y ALGORITMOS 1 - PRIMER SEMESTRE DE 2019
OBLIGATORIO: MINI FILESYSTEM

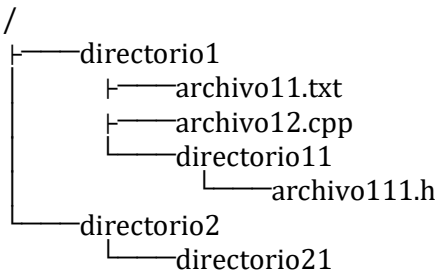
Se desea construir un *simulador* del administrador de archivos y directorios – file system – de un sistema operativo, el cual debe implementar un conjunto de comandos básicos de manejo de archivos y directorios, con sintaxis y comportamientos similares al Sistema Operativo UNIX.

MINI FILESYSTEM

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 el carácter reservado "/" (barra). Excepto por este directorio, los demás directorios deberán tener al menos 1 carácter alfanumérico y no podrán contener los caracteres '/' (barra) ni el '.' (punto).

Los archivos que administrará esta estructura serán de texto. Un archivo es identificado por un *nombre* de por lo menos un carácter, un punto y una *extensión* de por lo menos un carácter. Los caracteres válidos tanto para el nombre como para la extensión serán todos menos '/' (barra) y '.' (punto).

Ejemplo:



"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"

CONSIDERACIONES GENERALES

1. Se deben respetar mayúsculas y minúsculas para comparar cadenas de caracteres. Por ejemplo, los archivos planilla.xls, Planilla.Xls y PLANILLA.XLS son considerados distintos.
2. Un directorio puede contener un número ilimitado de subdirectorios y archivos.
3. Los archivos tienen un atributo de visibilidad, que permite ocultarlos en los listados que no especifiquen lo contrario.

TIPOS DE DATOS

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

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

El sistema debe permitir realizar las funciones que se detallan más adelante. Cada función (menos constructor, destructor) debe retornar uno de los siguientes tipos:

OK	Se debe retornar cuando la función ejecuta exitosamente.
ERROR	Se debe retornar cuando la función no puede ejecutar. Solo en este caso se debe imprimir el siguiente mensaje en la consola: *ERROR: mensaje*. Los mensajes posibles para cada operación del sistema serán indicados. En caso de producirse un error, la operación causante del error no tendrá efecto y la estructura que representa el filesystem permanecerá inalterada. Si la operación registra más de un error se emitirá un sólo mensaje de error correspondiente a alguno de los errores producidos.
NO_IMPLEMENTADA	Se debe retornar cuando la función no fue implementada.

Hay solamente dos casos en los cuales las funciones imprimen en la salida por consola:

- Cuando la función debe mostrar un mensaje porque hay un error.
- Cuando la función tiene como finalidad realizar un listado.

En ningún otro caso se mostrarán mensajes. Además, se deben respetar los formatos de las salidas que se muestran en la letra y en el ejemplo de uso (se entregará como proyecto Visual C++).

En ningún caso se aceptarán datos de entrada. El sistema desarrollado no será interactivo, sino que se limitará a ejecutar las operaciones definidas más adelante.

Se requiere implementar el TAD ListaPos en la clase ListaPosImp.

OPERACIONES

CREAR FILESYSTEM – CONSTRUCTOR (MAX_RECUPERAR)

Descripción: Inicializa el filesystem 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. Recibe el máximo número de archivos que se pueden recuperar en el filesystem (ver función undelete).

Ejemplo:

```
Filesystem *fs = new Filesystem(5);
fs->mkdir("/directorio1");
fs->mkdir("/directorio1/directorio11");
fs->dir("/", "");
```

Salida:

```
/directorio1
/directorio1/directorio11
```

Nota: las funciones *mkdir y dir* se desarrollan más adelante.

```
Filesystem (unsigned int MAX_RECUPERAR);
```

DESTRUIR FILESYSTEM – DESTRUCTOR ( )

Descripción: Debe liberar toda la memoria ocupada y asociada al objeto filesystem.

Ejemplo:

```
Filesystem *fs = new Filesystem(8);
...
delete fs;
```

```
~Filesystem ();
```

OPERACIONES RELATIVAS A LOS DIRECTORIOS

CREAR DIRECTORIO – MKDIR (RUTA)

Descripción: Este comando *crea* un nuevo subdirectorio vacío en la ruta especificada.

Ejemplo:

/
└———directorio1

fs->mkdir("/directorio1/directorio11");

/
└———directorio1
└———directorio11

```
TipoRetorno mkdir (char *rutaDirectorio);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: Ya existe un subdirectorio con el mismo nombre en esa ruta.
ERROR: No se puede volver a crear el directorio raíz.

ELIMINAR DIRECTORIO – RMDIR (RUTA)

Descripción: Este comando *elimina* un subdirectorio en la ruta especificada sin importar si está oculto o no.

Como convenciones, establecemos que:

- Al eliminar un directorio se eliminarán todos sus subdirectorios junto con los archivos contenidos en cada uno de ellos.
- Si se especifica el directorio RAIZ entonces se deberán eliminar todos los directorios y archivos del file system excepto el directorio RAIZ quedando el file system vacío.

Ejemplo:

/
└———directorio1
└———directorio11

fs->rmdir("/directorio1/directorio11");

/
└———directorio1

```
TipoRetorno rmdir (char *rutaDirectorio);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.

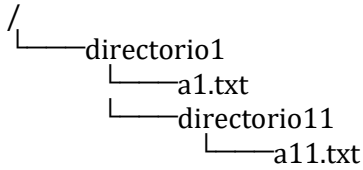
COPIAR DIRECTORIO – COPYDIR (RUTAORIGEN, RUTADESTINO)

Descripción: Este comando *copia* un directorio de una ubicación a otra.

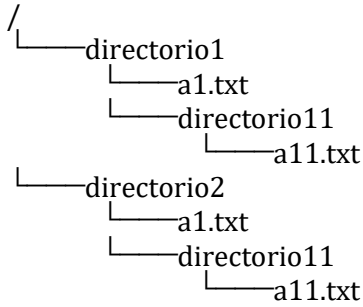
Como convenciones, establecemos que:

- Al copiar un directorio se copiarán todos archivos, los subdirectorios y los archivos que contengan.
- Los directorios y archivos copiados deben ser completamente independientes (no comparten memoria).
- El directorio de destino no puede ser un subdirectorio del origen.
- Todos los atributos de los directorios y los archivos se deben mantener.
- El directorio destino no debe existir.

Ejemplo:



fs->copydir("/directorio1", "/directorio2");




```
TipoRetorno copydir (char *rutaOrigen, char *rutaDestino)
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta origen.
ERROR: Ya existe la ruta destino.
ERROR: La ruta destino no puede ser un subdirectorio de origen.
ERROR: No se encuentra el padre de la ruta destino.

MOSTRAR DIRECTORIOS – DIR (RUTA, PARAMETRO)

Descripción: Este comando *muestra* el contenido del directorio especificado, ya sean subdirectorios o archivos. Acepta el parámetro -H en la forma que veremos más adelante.

Formato: primero se imprime la ruta completa del directorio especificado, desde el directorio RAIZ. Luego se listan los archivos del directorio especificado y posteriormente el contenido de cada uno de los subdirectorios siguiendo el mismo procedimiento (recursivamente). Tanto el listado de archivos como el de directorios deben ser realizados en orden alfabético (primero el menor) y debe incluirse en cada caso la ruta completa desde el directorio RAIZ.

Ejemplo: supongamos que contamos con la siguiente estructura:

```
/
├──archivo1.txt
├──directorio1
│       └──directorio11
└──directorio2
        ├──directorio20
        └──directorio21
                ├──archivo211.txt (oculto)
                ├──archivo212.txt
                ├──archivo213.cpp
                └──directorio211
                        ├──archivo2111.txt (oculto)
                        └──archivo2112.h
```

La ejecución del comando sin parámetros debería generar la siguiente salida por pantalla, mostrando el contenido del directorio especificado. Solamente se deberán mostrar los archivos y directorios que no se encuentren ocultos.

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

DIR /

```
/
/archivo1.txt
/directorio1
/directorio1/directorio11
/directorio2
/directorio2/directorio20
/directorio2/directorio21
/directorio2/directorio21/archivo212.txt
/directorio2/directorio21/archivo213.cpp
/directorio2/directorio21/directorio211
/directorio2/directorio21/directorio211/archivo2112.h
```

Ejemplo: suponiendo la estructura anterior, cuando no hay archivos ni directorios para mostrar se debe imprimir:

DIR /directorio2/directorio20

```
/directorio2/directorio20
No contiene archivos ni directorios.
```

Como convenciones establecemos que:

- El comando DIR realizará un listado de la información contenida en el directorio especificado.
- En primer lugar se imprimirá la ruta completa del directorio especificado. Luego, serán listados los archivos, en orden alfabético (primero el menor) y posteriormente los directorios también en orden alfabético.

Parámetro:

- **“”** Muestra solamente los archivos y directorios que no están ocultos.
- **“-H”** Muestra todos los archivos y directorios independientemente de si están ocultos o no. Si se imprime un archivo oculto, luego del nombre completo irá el string “(H)”, separado por 1 espacio en blanco.

Ejemplo: Si imprimimos con el parámetro –H:

DIR /directorio2/directorio21 -H

```
/directorio2/directorio21
/directorio2/directorio21/archivo211.txt (H)
/directorio2/directorio21/archivo212.txt
/directorio2/directorio21/archivo213.cpp
/directorio2/directorio21/directorio211
/directorio2/directorio21/directorio211/archivo2111.txt (H)
/directorio2/directorio21/directorio111/archivo2112.h
```

Nota: El formato de salida debe ser exactamente igual al de los ejemplos anteriores. Si se trata de un archivo oculto, luego del nombre completo irá el string “(H)”, separado por 1 espacio en blanco.

```
TipoRetorno dir (char *rutaDirectorio, char *parametro);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: Parametro desconocido.

OPERACIONES RELATIVAS A LOS ARCHIVOS

CREAR ARCHIVO – CREATEFILE (RUTA)

Descripción: Este comando *crea* un nuevo archivo en la ruta especificada.

Como convenciones establecemos que en su creación:

- Los archivos no estarán ocultos.
- 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 especificado (case-sensitive).

Ejemplo (Suponiendo que contamos con la siguiente estructura):

```
/
├──archivo1.txt
└──directorio1
      └──directorio11
            ├──archivo111.txt
            ├──archivo112.txt
            ├──archivo113.cpp
            └──directorio111
                  └──archivo.ppt
```

Si ejecutamos los siguientes comandos el resultado debería ser:

fs->createfile("/directorio1/directorio11/archivo2.txt");
fs->dir("/directorio1/directorio11");

```
/directorio1/directorio11
/directorio1/directorio11/archivo111.txt
/directorio1/directorio11/archivo112.txt
/directorio1/directorio11/archivo113.txt
/directorio1/directorio11/archivo2.txt
/directorio1/directorio11/directorio111
/directorio1/directorio11/directorio111/archivo.ppt
```

```
TipoRetorno createfile (char *rutaArchivo);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: Ya existe un archivo con el mismo nombre en esa ruta.

ELIMINAR ARCHIVO – DELETE (RUTA)

Descripción: Este comando *elimina* un archivo de la ruta especificada sin importar si está oculto o no.

Ejemplo (si deseamos eliminar el archivo creado anteriormente):

fs->delete("/directorio1/directorio11/archivo2.txt");
fs->dir("/directorio1/directorio11");

```
/directorio1/directorio11
/directorio1/directorio11/archivo111.txt
/directorio1/directorio11/archivo112.txt
/directorio1/directorio11/archivo113.txt
/directorio1/directorio11/directorio111
/directorio1/directorio11/directorio111/archivo.ppt
```

```
TipoRetorno delete (char *rutaArchivo);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: No existe un archivo con ese nombre en esa ruta.

MODIFICAR PARÁMETROS – ATTRIB (RUTA, PARAMETRO)

Descripción: Este comando *cambia* los atributos de un archivo perteneciente al directorio especificado.

El parámetro puede tomar los valores "+H" y "-H". El primero oculta el archivo indicado en la ruta y el segundo lo vuelve visible.

Por ejemplo, retomando la estructura del último ejemplo y considerando que el directorio especificado es "/directorio1/directorio11", si deseamos que el archivo "archivo112.txt" sea oculto, entonces:

ATTRIB /directorio1/directorio11/archivo112.txt +H

DIR /directorio1/directorio11 -H

```
/directorio1/directorio11
/directorio1/directorio11/archivo111.txt
/directorio1/directorio11/archivo112.txt (H)
/directorio1/directorio11/archivo113.txt
/directorio1/directorio11/directorio111
/directorio1/directorio11/directorio111/archivo.ppt
```

De la misma manera, podremos restablecerlo a no oculto ejecutando el comando inverso:

ATTRIB /directorio1/directorio11/archivo112.txt -H

DIR /directorio1/directorio11

```
/directorio1/directorio11
/directorio1/directorio11/archivo111.txt
/directorio1/directorio11/archivo112.txt
/directorio1/directorio11/archivo113.txt
/directorio1/directorio11/directorio111
/directorio1/directorio11/directorio111/archivo.ppt
```

Si el archivo ya está oculto y se le vuelve a poner el atributo oculto la funcion simplemente no tiene efecto, no retorna error. Lo mismo si se desea hacer visible.

Ejemplo:

ATTRIB /directorio1/directorio11/archivo112.txt +H

ATTRIB /directorio1/directorio11/archivo112.txt +H

```
TipoRetorno attrib (char *rutaArchivo, char *parametro);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: No existe un archivo con ese nombre en esa ruta.
ERROR: Parametro desconocido. *(SI SE INGRESA UN VALOR DIFERENTE DE -H o +H)*

OPERACIONES RELATIVAS A LA MODIFICACIÓN DE ARCHIVOS

Se desea implementar, además, algunos comandos básicos que administren el contenido de los archivos del sistema. Recordamos que los archivos serán de texto, en donde un texto se define como una secuencia de líneas que contienen caracteres.

Los comandos a implementar son los siguientes:

INSERTAR TEXTO – INSERTTEXT (RUTAARCHIVO, LINEA, POSICION, TEXTO)

Descripción: *Agrega* un texto en la línea y posición del archivo RutaArchivo. Si el número de línea no existe, se deben insertar líneas vacías hasta llegar a la nueva línea. Si existen caracteres en la posición marcada entonces el nuevo texto se insertará corriendo los caracteres en esa posición hacia adelante. En caso de que no haya caracteres que permitan llegar a la posición se deberán insertar espacios en blanco. El número de línea y la posición comienzan en 1.

No hay límite en la cantidad de caracteres por línea ni en la cantidad de líneas. Todos los archivos se pueden escribir estén ocultos o no.

Continuando con el ejemplo anterior, si el directorio especificado es "/directorio1/directorio11", el comportamiento esperado al ejecutar los siguientes comandos es:

INSERTTEXT/directorio1/directorio11/archivo112.txt 1 3 "Los peces del estanque"

TYPE /directorio1/directorio11/archivo112.txt

(TYPE, que muestra el contenido de un archivo, se describe más adelante)

```
/directorio1/directorio11/archivo112.txt

1:   Los peces del estanque                (empieza con 2 espacios en blanco)
```

INSERTTEXT/directorio1/directorio11/archivo112.txt 1 7 "cinco "

TYPE /directorio1/directorio11/archivo112.txt

```
/directorio1/directorio11/archivo112.txt

1:   Los cinco peces del estanque          (empieza con 2 espacios en blanco)
```

```
TipoRetorno insertText (char *rutaArchivo, unsigned int nroLinea, unsigned int
posLinea, char *texto);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.

ERROR: No existe un archivo con ese nombre en esa ruta.
ERROR: La linea cero no es valida.
ERROR: La posicion cero no es valida.

Eliminar texto – deletetext (ruta, linea, posicion, k)

Descripción: *Elimina* los a lo sumo K caracteres, comenzando en la línea y posición especificada (inclusive). Se deben respetar las siguientes afirmaciones:

- Si la línea no existe entonces se produce un error.
- Si no hay caracteres en esa posición la operación no tiene efecto (no hace nada).
- Si en la línea no hay K caracteres para borrar no intenta borrar caracteres de la siguiente línea. Borra solo los que puede de esa línea.
- Si al ejecutar el comando se borran todos los caracteres de una línea, se elimina la línea.

Todos los archivos se pueden escribir indistintamente de si están ocultos o no.

Sobre el ejemplo previo,

DELETETEXT/directorio1/directorio11/archivo112.txt 1 18 20
TYPE /directorio1/directorio11/archivo112.txt

```
/directorio1/directorio11/archivo112.txt

1:   Los cinco peces
```

DELETETEXT/directorio1/directorio11/archivo112.txt 1 7 6
TYPE /directorio1/directorio11/archivo112.txt

```
/directorio1/directorio11/archivo112.txt

1:   Los peces
```

DELETETEXT/directorio1/directorio11/archivo112.txt 1 1 50
TYPE /directorio1/directorio11/archivo112.txt

```
El archivo no posee contenido
```

INSERTTEXT/directorio1/directorio11/archivo112.txt 3 1 “Ayer vi una gaviota”
INSERTTEXT/directorio1/directorio11/archivo112.txt 1 1 “Que dia hermoso”
TYPE /directorio1/directorio11/archivo112.txt

```
/directorio1/directorio11/archivo112.txt

1: Que dia hermoso
2:
3: Ayer vi una gaviota
```

DELETETEXT/directorio1/directorio11/archivo112.txt 1 1 100
TYPE /directorio1/directorio11/archivo112.txt

```
/directorio1/directorio11/archivo112.txt

1:
2: Ayer vi una gaviota
```

`TipoRetorno deleteText (char *rutaArchivo, unsigned int nroLinea, unsigned int posLinea, unsigned int k);`

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: No existe un archivo con ese nombre en esa ruta.
ERROR: La linea cero no es valida.
ERROR: La linea no existe.
ERROR: La posicion cero no es valida.

MOSTRAR ARCHIVO – TYPE (RUTA)

Descripción: Este comando *imprime* el contenido del archivo parámetro.

Se debe imprimir primero la ruta del archivo y luego de una línea en blanco todas las líneas del archivo precedidas por el número de línea, el carácter ':' y un espacio en blanco. El formato correcto se puede ver en los ejemplos anteriores.

Todos los archivos se pueden mostrar indistintamente de si están ocultos o no.

Salida si el archivo no posee contenido:

```
/directorio2/directorio21/archivo213.txt


El archivo no posee contenido
```

Salida si el archivo está oculto:

```
/directorio2/directorio21/archivo213.txt


El archivo esta oculto
```

```
TipoRetorno type (char *rutaArchivo);
```

Errores posibles:
ERROR: La ruta no comienza con /.
ERROR: No se encuentra la ruta.
ERROR: No existe un archivo con ese nombre en esa ruta.

Nota: No es un error el caso de un archivo con contenido vacío o que el archivo se encuentre oculto. En esos casos deberá mostrar el mensaje indicado.

RECUPERAR ARCHIVO ELIMINADO – UNDELETE ()

Descripción: Este comando recupera el último archivo eliminado del filesystem.

Como convenciones, establecemos que:

- Se guardan a lo sumo MAX_UNDELETE archivos para recuperar. Los archivos borrados más antiguos no se pueden recuperar. Esta constante se recibe en el constructor del filesystem.
- El archivo debe ser recuperado en la misma ruta en la que se encontraba al ser eliminado. Si la ruta no existe, el archivo no se recupera y se elimina definitivamente (dando lugar a recuperar el siguiente archivo).
- Si hay un archivo con el mismo nombre en la ruta, el archivo no se recupera y se elimina definitivamente.
- El archivo recuperado debe mantener sus atributos.
- Los archivos se pueden recuperar indistintamente de si están ocultos o no y se recuperan con la misma visibilidad que cuando fueron eliminados.

```
TipoRetorno undelete ()
```

Errores posibles:
ERROR: No hay archivos para recuperar.

Nota: No es un error el caso de un archivo que no se puede recuperar. Si no se puede recuperar se debe eliminar definitivamente.

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.

Tipo 1	**Tipo 2**
`Constructor`	`rmdir`
`Destructor`	`copydir`
`mkdir`	`delete`
`dir`	`attrib`
`createFile`	`deleteText`
`insertText`	`undelete`
`type`	

Se requiere implementar el TAD ListaPos en la clase ListaPosImp.

SOBRE LA ENTREGA:

A fin de construir el Mini FileSystem se pide:

1. Definición y desarrollo de los TAD involucrados, siguiendo la metodología basada en clases abstractas y concretas vista en el curso, y usando C++ como lenguaje de programación.
2. Implementación de la clase interfaz del sistema que será provista por la cátedra. El código de todo el sistema debe estar desarrollado y comentado de acuerdo con las prácticas presentadas en el curso. Incluir las pre y postcondiciones de las operaciones de los TAD's.
3. **Se sugiere incluir casos de prueba propios del sistema, adicionales a los dados por la cátedra. Estos podrían ser relevantes durante la defensa del proyecto.**

4. Memoria de las tareas desarrolladas por cada integrante del grupo. Si desarrollaron todo el obligatorio en conjunto alcanza a aclararlo en palabras.

Operación	Fue Implementada	Estado de entrega	Desarrollada por		COMENTARIOS
			ESTUDIANTE 1	ESTUDIANTE 2	
	SI/NO	1. no compila 2. compila pero no ejecuta correctamente 3. ejecuta con algunos errores que se explican en los comentarios 4. ejecuta sin errores	Se deja en blanco en caso de que el estudiante no haya participado del desarrollo y en caso contrario se incluye un breve comentario acerca de en qué forma participó	Se deja en blanco en caso de que el estudiante no haya participado del desarrollo y en caso contrario se incluye un breve comentario acerca de en qué forma participó	Cualquier observación que se considere pertinente acerca de la operación

5. Informe breve fundamentando decisiones de diseño y desarrollo. Este informe debe contener el diseño del sistema: una justificación de los TAD's utilizados para resolver el problema (incluir diagrama de clases o de estructuras) y las implementaciones escogidas. Mencionar dos de los algoritmos más relevantes usados. El informe deberá contener una "***foto***" de cada integrante del grupo, junto con sus datos. En la página web del curso (sección obligatorios) estará disponible un documento con la estructura sugerida del informe solicitado.

Se debe entregar por gestión un zip que contenga:

- La carpeta del proyecto conteniendo todos los archivos utilizados (código fuente y archivos de proyecto Visual Studio)
- Fotos de ambos integrantes en formato jpg o png (Solo de la cara y que sea una foto reciente, no copia de documentos).
- Informe en formato pdf. El archivo debe llamarse informe.pdf (no incluir el código fuente como parte del informe).

NOTAS:

- La presente entrega vale **30 puntos**.
- La cátedra proporcionará una clase interfaz para el sistema, un diseño del mismo basado en TADs, al menos un ejemplo de uso y la especificación de un TAD Lista, que se sugiere fuertemente usar en el obligatorio.
- La documentación del sistema tendrá puntaje, al igual que la instancia de defensa final del proyecto.
- Toda la información concerniente al obligatorio (letra, archivos de la interfaz, casos de prueba, convenciones, etc.) estará publicada en el sitio web del curso y será actualizada periódicamente, en caso de ser necesario.