# Algoritmos y Programación II (75.41)
# Trabajo Práctico Nº1

19 de septiembre de 2014

## 1. Introducción

La ALS Association (asociación abocada al desarrollo e investigación de posibles curas para la Esclerosis Lateral Amiotrófica) se encuentra totalmente sobrepasada por el enorme éxito que ha tenido su reciente campaña conocida como Ice Bucket Challenge.

Por este motivo, han decidido solicitar la ayuda de los alumnos de Algoritmos II para desarrollar un sistema que les permita administrar y mantener un registro de la realización de la campaña.

La campaña consiste en difundir información sobre la enfermedad a través de videos en los que cada participante puede donar dinero o arrojarse un balde de agua helada frente a las cámaras y también nominar a otras personas a realizar el desafío.

En virtud de algunos inconvenientes por disputas de cartel entre ciertos participantes del mundo del espectáculo, la asociación informó que la decisión final sobre tirarse el balde o no (que es la que generalmente queda registrada en video), deberá ser ingresada al sistema por orden de aparición.

## 2. Consigna

Implementar en C el sistema de gestión del Ice Bucket Challenge, que debe proveer las siguientes operaciones:

- Donar
- Nominar
- Tirar Balde

### 2.1. Protocolo

Este programa formará parte de un sistema automatizado. En lugar de presentar un menú con opciones en la pantalla, se esperará que el programa

respete un *protocolo de comunicación*, en el cual se reciben comandos por la entrada estándar (`stdin`). Cada comando recibido debe ser procesado y su resultado debe ser escrito en la salida estándar (`stdout`).

Cada línea de la entrada corresponde a un comando, y todos los comandos respetan un determinado formato, tal como se muestra a continuación:

```
participante comando parametro1 parametro2 parametro3 ... ↩
    ↪parametroN
```

## 2.2. Comandos

### 2.2.1. Comando: `donar`

**Formato:** `participante donar cantidad`

**Descripción:** El participante dona una cantidad de dinero para la campaña. Si el participante es nuevo, se lo registra. Si ya existía, se incorpora la donación a su cuenta.

**Parámetros:**

`cantidad`**:** Un número entero positivo mayor a 0 que representa la cantidad de dinero donada.

**Salida:** `OK`**:** en caso de no producirse error.

`ERROR1`**:** si la cantidad es menor o igual a 0.

**Ejemplo:**

```
Daniela donar 100
OK
Daniela donar 0
ERROR1
Daniela donar -10
ERROR1
Daniela donar 200
OK
```

### 2.2.2. Comando: `nominar`

**Formato:** `participante nominar nominado1 nominado2 ... nominadoN`

**Descripción:** Nomina a nuevos participantes a realizar el desafío. Si el nominado no existía en el sistema se lo registra. Si el participante ya existía en el sistema (por haber donado o haber sido nominado

anteriormente) devuelve error. Un participante puede nominar (una o más veces) sólo si ha realizado una donación. Ningún participante puede ser nominado más de una vez. La cantidad de nominados es variable.

**Parámetros:**

`nominado1`**:** Nombre del primer nominado.

`nominado2`**:** Nombre del segundo nominado.

`nominadoN`**:** Nombre del nominado N.

**Salida:** Una línea por nominado

`OK nominado`**:** en caso de no producirse error.

`ERROR2`**:** si el PARTICIPANTE que pretende nominar a otros, no existe en el sistema (nunca donó ni fue nominado).

`ERROR3`**:** si el PARTICIPANTE que pretende nominar existe pero no ha realizado donaciones.

`ERROR4 nominado`**:** si alguno de los nominados ya estaba registrado (por haber donado o por haber sido nominado anteriormente, incluso después de tirar_balde).

**Ejemplo:**

```
Daniela nominar Martín Pablo
OK Martín
OK Pablo
Diego nominar Ezequiel
ERROR2
Martín nominar Ezequiel
ERROR3
Daniela nominar Pablo
ERROR4 Pablo
```

### 2.2.3. Comando: `tirar_balde`

**Formato:** `participante tirar_balde decisión`

**Descripción:** El participante confirma su decisión de tirarse el balde por sí o por no. Para poder informar su decisión el participante debe ser el siguiente en el orden de aparición (según se fueron registrando en el sistema, al donar o al ser nominados). Por lo tanto, cada participante sólo puede informar su decisión de tirarse el balde una sola vez (en el momento en el que le toque según el orden de aparición).

**Parámetros:**

`decisión:` [si/no]

**Salida:** `OK [DONACIONES] TOTAL: N` Si el comando es exitoso, se deben listar las donaciones en el orden inverso al que fueron realizadas (la más reciente primero) y luego el total donado

`ERROR5:` si el participante no es el siguiente en el orden de aparición.

`ERROR6:` si el participante no donó nada y además decide no tirarse el balde.

**Ejemplo:**

```
Daniela tirar_balde si
OK [200,100] TOTAL: 300
Pablo tirar_balde si
ERROR5
Martin tirar_balde no
ERROR6
Martin tirar_balde si
OK [] TOTAL: 0
```

# 3. Pruebas

Junto con la especificación se provee de **pruebas automáticas para el programa completo**. Estas pruebas serán de utilidad para revisar que el programa cumpla con la especificación del protocolo de entrada/salida.

Para la aprobación del Trabajo Práctico es requisito que el programa implementado pase todas las pruebas.

## 3.1. Ejecución de las pruebas

Una vez descomprimido el archivo zip con las pruebas[1], se debe efectuar los siguientes pasos para correr las pruebas:

1. Compilar el programa (supongamos que se llama `tp1`)
2. Ejecutar[2]

```
$ bash pruebas/correr-pruebas.sh ./tp1
```

[1] Puede descomprimirse en cualquier lugar; para el ejemplo suponemos que se guardó en la misma carpeta que el TP. Es decir, el ejecutable `tp1` quedaría al mismo nivel que la carpeta `pruebas` (que contiene el archivo `correr-pruebas.py`).

[2] Para correr las pruebas es necesario disponer de un entorno con línea de comandos Bash y herramientas GNU. En Linux seguramente no sea necesario instalar nada.

Cada una de las pruebas está especificada en un archivo con extensión `.test` dentro de la carpeta de pruebas. Por ejemplo:

```
pruebas/
 \- correr-pruebas.sh
 \- prueba1.test
 \- prueba2.test
```

El script `correr-pruebas.sh` ejecutará el programa una vez por cada prueba, pasándole en la entrada estándar los comandos especificados en la prueba. Luego verificará que la salida estándar del programa sea exactamente igual a la esperada según el protocolo.

# 4. Criterios de aprobación

A continuación describimos criterios y lineamientos que deben respetarse en el desarrollo del trabajo.

## 4.1. Utilización de estructuras de datos

Para la realización de este trabajo es necesario utilizar las estructuras de datos vistas en clase (vector dinámico, pila, cola, lista), además de crear las estructuras adicionales que se consideren necesarias.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas.

## 4.2. Programa

El programa debe cumplir los siguientes requerimientos:

- Debe estar adecuadamente estructurado y modularizado, utilizando funciones definidas de la forma más genérica posible, sin caer en lo trivial.
- El código debe ser claro y legible.
- El código debe estar comentado y las funciones definidas, adecuadamente documentadas.
- El programa debe compilar sin advertencias ni mensajes de error[3], debe correr sin pérdidas de memoria, uso de valores sin inicializar, o errores en general. Es decir que, el programa debe correr en valgrind sin errores.
- Además, claro, debe satisfacer la especificación de la consigna y pasar todas las pruebas automáticas.

[3] `-Wall -pedantic -std=c99`

### 4.3. Informe

El informe deberá consistir de las siguientes partes:

- **Carátula** con la información del alumno/a y el ayudante asignado (en caso de saberlo de antemano).
- **Análisis y diseño**: Describir la solución elegida para resolver cada problema propuesto, y cómo se lleva a cabo. En particular, mencionar cómo es el flujo del programa, qué algoritmos y estructuras de datos se utilizan y por qué, y cuál es el orden de ejecución en tiempo y espacio de cada operación.
- **Implementación**: Incluir aquí *todo* el código fuente utilizado (en formato `monoespaciado`, para facilitar su lectura).
- También *opcionalmente,* toda explicación adicional que consideren necesaria, referencias utilizadas, dificultades encontradas, cambios o mejoras que se podrían hacer a futuro y conclusiones.

El informe debe estar lo más completo posible, con presentación y formato adecuados. Por ejemplo, este enunciado cumple con los requerimientos de un informe bien presentado.

## 5. Entrega

El trabajo consiste en:

- El informe impreso.
- El informe digital, en formato `.pdf`
- Una versión digital de **todos** los archivos de código fuente, separados del informe, en un archivo comprimido (`.zip` o `.tar.gz`).

Los dos últimos deben enviarse a la dirección `tps.7541rw@gmail.com`, colocando como asunto:

```
TP1 - Padrón - Apellido
```

Se aclara que por código fuente se entiende todos los archivos `.h` y `.c`, el archivo `Makefile` para poder compilar, y todos los archivos adicionales que sean necesarios para ejecutar el programa. No deben entregarse nunca archivos `.o` u otros archivos compilados.

El informe impreso debe entregarse en clase. El plazo de entrega vence el **Viernes 3 de Octubre de 2014**.