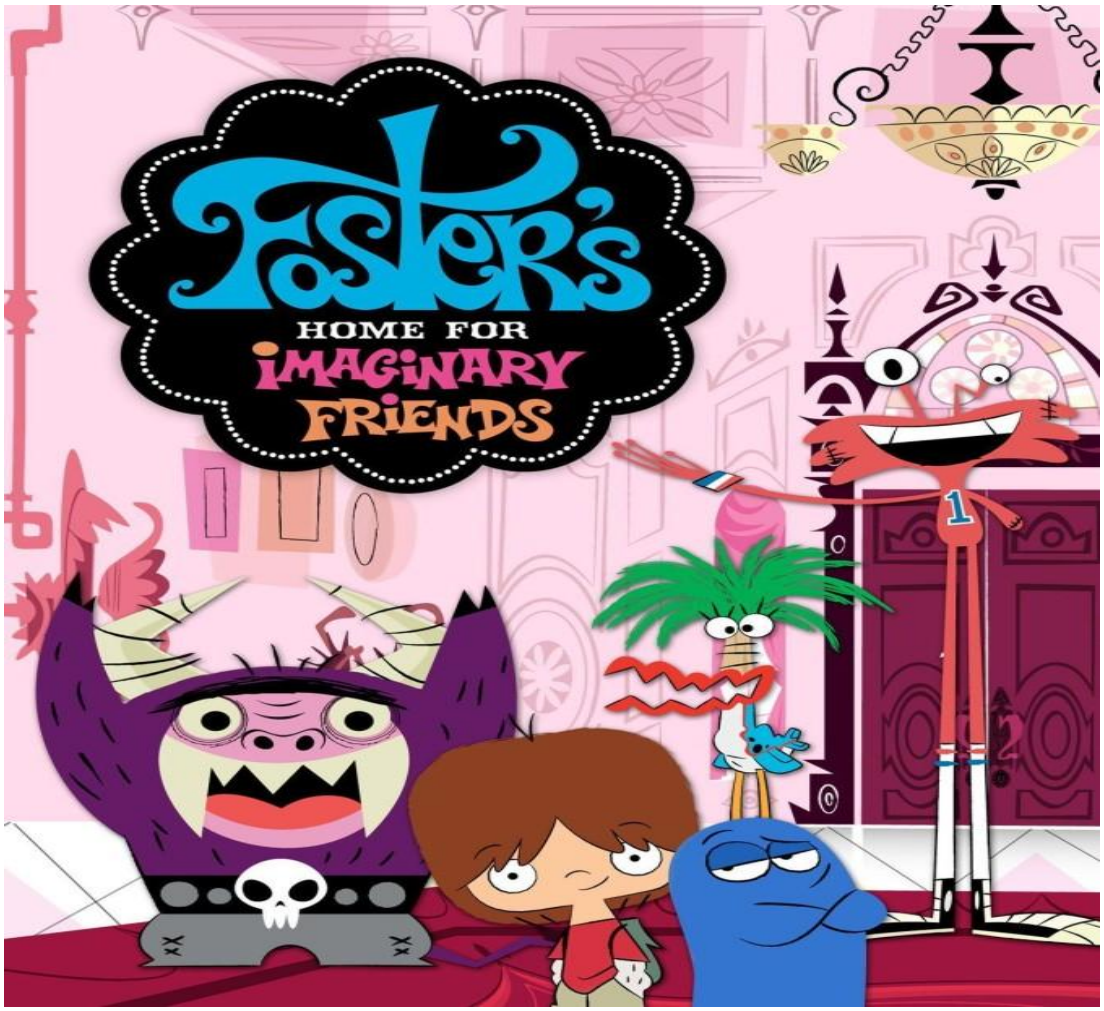
Algoritmos y Programación I

Curso Camejo

.UBAfiuba
FACULTAD DE INGENIERÍA

TP Tercera Parte

La Sombra de Mac



Fecha Presentación	25/11/2021
Fecha Entrega	9/12/2021

1. Introducción

Mac es un niño de ocho años con gran inteligencia y creatividad. Él es una persona sensata y moral, y puede ser a través de Bloo, su amigo imaginario, que hace y dice todas las cosas que quiere pero no puede. Por lo tanto, Bloo existe como un desafío a la moralidad de Mac. Pasa una gran porción de sus días involucrándose en sus travesuras dentro y fuera de Foster's. La **Mansión Foster** para amigos imaginarios es un inmenso orfanato con estilo victoriano al que los amigos imaginarios se van a vivir cuando ya no pueden pertenecerle a sus creadores. Ahí se muda Bloo, y Mac lo visita todos los días para asegurarse de que Bloo no sea adoptado.

Bloo es el amigo imaginario de Mac, a menudo egoísta, rebelde, inquieto, arrogante o en busca de atención. No obstante, puede cambiar su actitud en lealtad a su creador. Su cuerpo humanoide completamente azul es comparable a la de un guante de cocina o un fantasma del videojuego Pac-Man. A menudo, tiene una tendencia a aceptar ideas escandalosas como un hecho para explicar sucesos aparentemente mundanos, y somete a otros a su voluntad, yendo tan lejos como para hacerlos ir en contra de sus estándares morales. Bloo también muestra pasión por la pelota de pádel, aunque nunca logra hacer que la pelota golpee la paleta, a lo que insiste en que todas sus paletas están rotas.

2. Objetivo

El presente trabajo práctico tiene como objetivo evaluar a los alumnos en aspectos fundamentales de la programación. Estos aspectos son:

- Validación de datos ingresados por el usuario.
- Diseño y desarrollo de funcionalidades de una biblioteca con un contrato preestablecido.
- El correcto uso de estructuras de control.
- Tipos de dato simples y estructurados.
- Buenas prácticas de programación.
- Modularización.
- Manejo de archivos en C.
- Línea de comandos.

3. Enunciado

Ya ayudamos a Mac y a Bloo a volver de la mansión luego de haber pasado una tarde explorando la ciudad.

Pero, ahora desde La Mansión Foster, debido a que recibieron mucha ayuda, nos piden empezar a llevar registro de todas las partidas que se fueron jugando, y así recompensar a quienes más hayan ayudado para que ellos puedan volver a la mansión. Para esto se usará una estructura del tipo partida_t, la cual se especificará más adelante.

Podrás ayudarlos una vez más?

4. Especificaciones

4.1. Estructuras

Nos manejaremos con una única estructura, **partida_t**, la cual es:

```c
#include <stdbool.h>

#define MAX_NOMBRE 100

typedef struct partida {
    char jugador[MAX_NOMBRE];
    int nivel_llegado;
    int puntos;
    int vidas restantes;
    bool gano;
} partida_t;
```

4.2. Archivos

Todos los archivos con los que se van a trabajar tendrán el mismo formato. Cada partida está representada en una línea del archivo de la forma:

```
Diego;3;10;2;Si
Juan carlos;3;26;0;No
Marta;2;15;0;No
Mirtha;3;10;3;Si
```

Observar que el campo booleano de la estructura en el archivo csv se representa como un string, se deberá manipular este último para definir el valor del booleano.

4.3. Funcionalidades

- Agregar partida.
- Eliminar partida.
- Ordenar partidas.
- Top partidas.
- Configuraciones.

4.3.1. Agregar Partida

Dado un archivo csv de partidas ordenado por nombre alfabéticamente, agregar una partida manteniendo el orden pre establecido.

Ejemplo: Si se tiene el archivo

```
Diego;3;10;2;Si
Juan carlos;3;26;0;No
Marta;2;15;0;No
Mirtha;3;10;3;Si
```

Y se quiere agregar una partida que tenga la forma:

```
Lionel;1;0;0;No
```

El archivo deberá quedar:

```
Diego;3;10;2;Si
Juan carlos;3;26;0;No
Lionel;1;0;0;No
Marta;2;15;0;No
Mirtha;3;10;3;Si
```

Este comando se ejecutará de la forma:

```
./mansion_foster agregar_partida <<nombre_archivo>>
```

Donde «nombre_archivo» representa el nombre del archivo al cual se ingresará la partida.

Observación: La información de la partida la ingresará por entrada estándar (a traves de scanfs) el usuario, y ahí se agregará al archivo ordenadamente.

4.3.2. Eliminar Partida

Dado un archivo csv de partidas ordenado por nombre alfabéticamente, eliminar una partida mantiendo el orden pre establecido.

Ejemplo: Si se tiene el archivo

```
Diego;3;10;2;Si
Juan carlos;3;26;0;No
Marta;2;15;0;No
Mirtha;3;10;3;Si
```

Y se quiere eliminar una partida que el jugador tenga el nombre: **Marta**.
El archivo deberá quedar:

```
Diego;3;10;2;Si
Juan carlos;3;26;0;No
Mirtha;3;10;3;Si
```

Este comando se correrá de la forma:

```
./mansion_foster eliminar_partida <<nombre_archivo>> <<nombre_jugador>>
```

Donde «nombre_archivo» representa el nombre del archivo donde se eliminará la partida. Y «nombre_jugador» será el argumento por el cual se buscará la partida que será eliminada. Si hay más de una partida con el mismo «nombre_jugador» se eliminará la primera que se encuentre.

4.3.3. Ordenar Partidas

Dado un archivo csv de partidas, se pide ordenarlo alfabéticamente por nombre. Aclaración: Se puede suponer que el archivo entra en memoria.
Este comando se correrá de la forma:

```
./mansion_foster ordenar_partidas <<nombre_archivo>>
```

Donde «nombre_archivo» representa el nombre del archivo se ordenará.

4.3.4. Top Partidas

Dado un archivo csv de partidas ordenado alfabéticamente por nombre, se pide imprimir por pantalla, y ademas, crear otro archivo csv, con las «cantidad_a_mostrar» partidas que hayan terminado con más puntos y ganado el juego.

Observación: En caso de que no haya al menos «cantidad_a_mostrar» partidas que cumplan con esto, se imprimen y se crea el archivo igualmente.

Ejemplo: Si se tiene el archivo

```
Diego;3;10;2;Si
Juan carlos;3;26;0;No
Marta;2;15;0;No
Mirtha;3;10;3;Si
```

Se deberá crear un archivo de la forma (a parte de imprimir por pantalla):

```
Diego;3;10;2;Si
Mirtha;3;10;3;Si
```

Este comando se correrá de la forma:

```
./mansion_foster top_partidas <<nombre_archivo>> <<cantidad_a_mostrar>>
```

Donde «nombre_archivo» representa el nombre del archivo se ordenará. Y donde «cantidad_a_mostrar» representa el número de partidas que se van a imprimir, y la cantidad de partidas que almacenará el nuevo archivo csv.

Observación: Queda a critero del alumno/a el hacer o no, más funciones y/o procedimientos para resolver los problemas presentados.

4.3.5. Configuraciones

Dado un archivo con ciertas configuraciones llamado config.txt, leerlo y aplicar las configuraciones determinadas para el juego.

Las configuraciones que podrán aparecer en el archivo son:

- Pozos
- Velas

- Interruptores
- Portales
- Monedas
- Escaleras
- Llaves

Y el formato será: N«numero_nivel»_«elemento»=«cantidad». Todo escrito en mayúsculas.
Ejemplo:

```
N1_POZOS=20
N1_VELAS=15
...
N2_POZOS=10
...
N3_POZOS=50
...
```

Aclaraciones: En caso de no existir el archivo, se jugará con las condiciones normales del juego.
Es de suma importancia remarcar que las configuraciones pueden venir en distinto órden, por lo que se deberá verificar la configuración que se esté leyendo.

Este comando se correrá de la forma:

```
./mansion_foster config_juego
```

5. Resultado esperado

Se espera que el trabajo creado cumpla con un propio manejo de archivos, las buenas prácticas de programación y todas las funciones y procedimientos pedidos funcionen acorde a lo solicitado, respetando las pre y post condiciones propuestas.

6. Compilación y Entrega

El trabajo debe poder ser compilado correctamente con la siguiente línea:

```
gcc *.c -o mansion_foster -std=c99 -Wall -Wconversion -Werror -lm
```

Por último debe ser entregado en la plataforma de corrección de trabajos prácticos **Chanutron2021** (patente pendiente), en la cual deberá tener la etiqueta **¡Exito!** significando que ha pasado las pruebas a las que la cátedra someterá al trabajo.

Para la entrega en **Chanutron2021** (patente pendiente), recuerde que deberá subir un archivo **zip** que contenga únicamente los archivos antes mencionados, sin carpetas internas ni otros archivos. De lo contrario, la entrega no será validada por la plataforma.

IMPORTANTE! Obtener la etiqueta **¡Exito!** en **Chanutron2021** (patente pendiente) no implica necesariamente haber aprobado el trabajo. El trabajo será corregido por un colaborador que verificará que se cumplan las buenas prácticas de programación.
Al poner *.c el compilador toma todos los .c de la carpeta y los compila. Por ende, no importa el nombre que le pongan a la biblioteca ni al programa.

7. Anexos

7.1. Argumentos por línea de comando

Para poder pasar parámetros a un programa a través de la línea de comandos nos valemos de la siguiente declaración de la función main:

```
int main(int argc, char *argv[]){..
```

Argc contiene la cantidad de argumentos recibidos por el programa, debemos considerar que siempre será el número de argumentos pasados más 1, ya que el primer argumento se reserva para contener el nombre del programa. **Argv** es un vector de strings que contiene los parámetros pasados en el mismo orden en que fueron escritos.

El siguiente programa muestra un ejemplo de cómo hacer uso de estos parámetros:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

        printf ("El programa recibió %i argumentos\n", argc);
        for(int i = 0; i < argc; i++) {
                printf("Parámetro %d: %s\n", i, argv[i]);
        }

        if (strcmp (argv[1], "saludar") == 0){
                printf ("Hola!\n");
        } else if (strcmp (argv[1], "sumar") == 0 && argc == 4){
                int sumando_1 =  atoi (argv [2]); // convierte el argumento argv[2] de string a
    numero
                int sumando_2 =  atoi (argv [3]);
                printf ("%i + %i = %i\n", sumando_1, sumando_2, sumando_1 + sumando_2);
        }
        return 0;
}
```

Referencias

Links:

https://fostershomeforimaginaryfriends.fandom.com/wiki/Mac
https://fostershomeforimaginaryfriends.fandom.com/wiki/Bloo