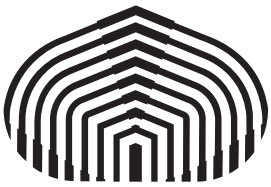

UNIVERSIDAD SIMÓN BOLÍVAR
DPTO. COMPUTACIÓN Y TEC. DE INF.
CI-3825 SISTEMAS DE OPERACIÓN I

# Proyecto 2: Los amigos de Ash Ketchum

*Estudiantes*
KAREN MARTINEZ
OLIVER BUENO
ALEJANDRO MENESES

*Profesor*
FERNANDO TORRES

17 de abril de 2023

# Contenido General

# 1. Introducción

Pokémon, una de las series de televisión más famosas de los últimos años, ha llegado a su fin después de más de 25 años. Su personaje principal Ash Ketchum, a lo largo de la serie, conoció a muchos personajes interesantes, algunos de los cuales dejo con la intención de verlos de nuevo. Sin embargo, a ¿cuántos personajes logró revisitar Ash en su aventura?, responder a esta pregunta no es tan sencillo dado la cantidad de personajes y la clasificación jerárquica en la que los fans de la serie la documentarón lo cual hace difícil llevar un recuento de ellos.

Por esta razón, se ha creado un programa que puede realizar diferentes consultas sobre los personajes según la región geográfica, especie, y tipo de apariciones.De modo que, en este informe, se describen la estructura del programa y las decisiones de diseño tomadas. Así como los problemas encontrados durante el desarrollo y algunas recomendaciones finales.

# 2. Estructura del programa y decisiones de diseño

## 2.1. Estructura

El programa se estructura en dos partes:

1. La parte del parser, que se encarga de leer y almacenar en una estructura adecuada los argumentos de los flags ingresados por el usuario desde la línea de comandos.
2. La parte del manejo de los directorios, donde se cuentan el número de apariciones de los archivos que cumplen los criterios ingresados por el usuario, y se muestran los resultados obtenidos por la salida.

## 2.2. Decisiones de diseño

La primera parte se esta en un archivo llamado `clic_parser.h`, donde se encuentran las siguientes funciones:

- `void handle_cli_args(int argc, char *argv[], options_t *opts)` se encarga del parseo de los argumentos desde la línea de comandos. Para la implementación de esta función se utilizó la librería getopt.h. Aquí se establece el comando `./fameChecker [-r <region>] [-s <species>] [-t <type>] [-c|--nocount] [-l|--list] [-z|--size] [name]` para la ejecución del programa.

  Donde:

  **`-r`:** indica que se debe limitar la búsqueda a una región.

  **`-s`:** indica la especie, “pokemon” para contar los archivos HTML en los directorios de pokémones que correspondan con los criterios de búsqueda o “trainer” para indicar que se debe contar solamente a los humanos

  **`-t`:** indica el tipo de apariciones: “main” para los personajes principales, “recurring” para los personajes recurrentes, “gym_leader” para los líderes de gimnasio, y “one_time” para los personajes que aparecen solo una vez.

  **`-c` ó `--nocount`:** indica si se debe mostrar el número de archivos encontrados. Por defecto, se muestra. Si el flag está presente, no se debe mostrar.

**-l ó --list:** indica si se deben dar los nombres de los archivos encontrados. Por defecto, no se muestran.

**-z ó --size:** indica si se debe mostrar el tamaño en kilobytes (1024 bytes) de los archivos. Por defecto, no se muestra. Si está incluido el flag -l, se muestra junto a cada archivo encontrado su tamaño.

**[name]:** permite restringir la búsqueda a archivos que comiencen con el nombre dado.

- `void print_usage(char *program_name)` muestra por la salida el usage del programa en caso de que la función anterior detecte algún error.

Se debe destacar que para la opción que indica que se muestre el tamaño de los archivos se cambio el flag `-s` por `-z` para evitar conflictos a la hora de parsear la entrada, ya que hay una opción con el mismo flag.

La segunda parte esta en un archivo llamado `directory_parser.h`, donde se encuentran las siguientes funciones:

- `void print_file(int list, int size, long int size_file, char* name_file)` muestra por la salida los archivos encontrados. Aquí se verifica si se deben mostrar tanto los archivos como el tamaño de los mismos, o si solo se debe mostrar alguno de ellos para lo cual se utiliza los parámetros de entrada.
- `inode_traversal(char* rut, int list, int size, char* name)` recorre los archivos de un directorio. Esta función se apoya en la anterior para cumplir el fin ya especificado.
- `traverse_directory_dfs(char* rut1, char* rut2, char* rut3, int nocount, int list, int size, char* name)` se encarga de recorrer los directorios de acuerdo con los comandos de entrada. Es una versión modificada del algoritmo *DFS* que realiza las siguientes verificaciones:
  1. Se ingresa la región, pero no el tipo de especie ni el tipo de aparición.
  2. Se ingresa el tipo de especie, pero no la región ni el tipo de aparición.
  3. Se ingresa el tipo de aparición, pero no la región ni el tipo de especie.
  4. Se ingresa la región y el tipo de especie, pero no el tipo de aparición.
  5. Se ingresa el tipo de especie y el tipo de aparición, pero no la región.
  6. Se ingresa la región y el tipo de aparición, pero no el tipo de especie.
  7. Se ingresa la región, el tipo de especie y el tipo de aparición.

Para cualquiera de las condiciones anteriores, se construye el path absoluto y se llama a la función recursivamente para seguir explorando. Además si no se especifica ninguna opción se recorre todo el arbol del directorio y se muestra el número de archivos encontrados.

Es importante destacar que en ningún momento se "cablea" alguna dirección, siempre se trabaja con direcciones relativas. Sin embargo, es imprescindible que el sistema jerárquico el que se ejecute el programa sea el siguiente:

```
/directorio
└── nivel 1
    └── nivel 2
        └── nivel 3
```

Figura 1: Jerarquía de archivos con la que se trabaja en el programa

Es decir, un sistema jerárquico de tres niveles, donde el primer nivel indica las regiones, el segundo nivel indica el tipo de especie y el tercer nivel indica el tipo de apariciones de cada personaje, así como su información resumida en un archivo .HTML. Ademas, el ejecutable del programa se coloca dentro del nivel 1.

# 3. Cambios en la implementación y resultados sorprendentes

Los cambios que se realización en la implementación, fueron la forma de acceso a los directorios del nivel 3. Inicialmente se creó una estructura que contenía tres arreglos dinámicos, cada uno con los nombres de los directorios según el nivel. Los nombres de cada directorio se obtenían con una versión modificada del algoritmo *DFS*, que en este caso sí recorría todo el árbol de directorios. Luego se intentó implementar el recorrido por los distintos arreglos de la estructura para obtener las rutas de los archivos. Sin embargo, se descartó esta implementación a favor de la versión modificada del algoritmo *DFS* con las verificaciones ya mencionadas, que consume menos recursos y es suficiente para obtener los archivos.

# 4. Conclusión

En resumen, el programa diseñado para la contabilización de los personajes de la serie Pokémon puede ser de gran utilidad para aquellos fanáticos que desean conocer más acerca de los personajes que aparecen en la serie, pues permiten llevar un seguimiento detallado de ellos.

El diseño del programa se enfocó en la simplicidad y eficiencia en la búsqueda y conteo de personajes, utilizando una estructura de dos partes: el parser y el manejo de los directorios.

## 4.1. Recomendaciones

- Se sugiere realizar una actualización periódica de la información contenida en los documentos HTML, con el fin de tener la información más actualizada y completa posible.
- También se podría incluir una función de ordenamiento para que los resultados de las consultas se muestren de manera ordenada y fácil de leer.
- Sería interesante implementar un sistema de búsqueda más avanzado que permita buscar personajes con características más específicas.
- También se podría mejorar la usabilidad del programa con una interfaz gráfica de usuario para hacerlo más accesible a personas que no estén familiarizadas con la línea de comandos.