DATA STRUCTURES AND ALGORITHMS II, 2022-2023

STUDENT: Albert, Naiara, Arnau y Helio
STUDENT: u214020, u215075, u214973, u214950
DATE OF SUBMISSION: 11-06-2023

ÍNDICE

INTRODUCCIÓN

Edahana es una innovadora red social la cual su principal objetivo es conectar a los personas a través de valoraciones de los usuarios.

Sus características principales son las siguientes:

Primeramente, encontramos los perfiles de usuarios los cuales se caracterizan por tener un perfil personalizado que ofrece una presentación única. Estos incluyen información como el nombre del usuario, una descripción breve, una lista de gustos y una valoración que indica la calidad de las interacciones y el contenido compartido por ese usuario. Esta valoración se asigna en función de las interacciones de otros usuarios que visitan el perfil y dejan comentarios. El sistema de valoración fomenta una comunidad donde se reconocen las contribuciones valiosas y se promueve la interacción respetuosa.

En segundo lugar, encontramos las publicaciones en las cuales los usuarios de la red pueden compartir sus pensamientos e ideas a través de mensajes cortos. Esta limitación en la longitud de los mensajes fomenta que se compartan de manera más directa y efectiva.

Finalmente, una característica destacada de Edahana es su sistema de valoración de usuarios, ya que ellos mismos tienen la capacidad de valorar sobre los perfiles de otros usuarios.

Cabe mencionar que la innovadora red social descrita anteriormente, podría tener algunas similitudes con el universo de la serie "Black Mirror" debido a que los usuarios también reciben una valoración basada en las interacciones y el contenido que comparten. Esto podría recordarnos episodios donde las personas son clasificadas según sus interacciones sociales.

2. OBJETIVOS DEL PROYECTO

2.1 Descripción general

Para empezar, cabe mencionar que nuestro trabajo se compone de cuatro apartados; en primer lugar, están las publicaciones que contienen la implementación de varias funciones relacionadas con la manipulación de las publicaciones que enviará cada usuario. En segundo lugar, los usuarios, donde se implementan funciones relacionadas con la información de cada perfil y las solicitudes de amistad. En tercer lugar, está el menú del programa, el cual sirve para acceder a los comandos y a las diferentes ventanas de la red social. Para acabar, encontramos el main el cual sirve para conseguir la ejecución del programa.

2.1.1 ¿Qué variables se utilizaron y con qué propósito?

Algunas de las variables que encontramos en las funciones del apartado “publicaciones” son las siguientes:

En la función insert_post:

- **publ**: Puntero a la estructura publicación donde se almacenará la nueva publicación.
- **user**: Puntero a la estructura User_data que contiene los datos del usuario que realiza la publicación.
- **post**: Cadena de caracteres que representa el contenido del post.

En la función push_post:

- **tl**: Puntero a la estructura timeline donde se agregarán las publicaciones.
- **user**: Puntero a la estructura User_data que contiene los datos del usuario que realiza la publicación.
- **post**: Cadena de caracteres que representa el contenido del post.

En la función pop_post:

- **tl**: Puntero a la estructura timeline de la cual se eliminará la última publicación.

  En la función bubblesort_dictionary:
- **dict**: Array de punteros a estructuras diccionario que se ordenará utilizando el algoritmo de ordenamiento de burbuja.
- **n**: Entero que representa el tamaño del array dict.

En la función show_top:

- **tl**: Puntero a la estructura timeline desde la cual se obtendrá la última publicación.

En la función contar_palabras:

- **tl**: Puntero a la estructura timeline desde la cual se contarán las palabras en las publicaciones.
- **actual**: Puntero a la estructura publicacion que recorre las publicaciones en la timeline.
- **dict**: Doble puntero a estructuras diccionario que se utilizará para almacenar las palabras y sus frecuencias.

En las funciones show_recent_posts_from_user y show_old_posts_from_user:

- **user**: Puntero a la estructura User_data que representa el usuario del cual se mostrarán las publicaciones.
- **tl**: Puntero a la estructura timeline desde la cual se obtendrán las publicaciones.
- **n**: Entero que indica el número de publicaciones que se mostrarán. Un valor de -1 indica que se mostrarán todas las publicaciones.

En el apartado “usuario” encontramos las siguientes variables de las siguientes funciones:

En la función insert_user:

- **miembro**: Un puntero a la estructura User_data donde se insertarán los datos del usuario.
- **nombre_usuario**: Una cadena de caracteres que representa el nombre de usuario.
- **correo**: Una cadena de caracteres que representa la dirección de correo electrónico.
- **contraseña**: Una cadena de caracteres que representa la contraseña.
- **ciudad**: Una cadena de caracteres que representa la ciudad.
- **año**: Un entero que representa el año de nacimiento.
- **num_usuario**: Un entero que representa el número de usuario.
- **gusto1 a gusto5**: Cadenas de caracteres que representan los gustos del usuario.
- **nota**: Un valor flotante que representa la calificación promedio del usuario.
- **nota_max**: Un valor flotante que representa la calificación máxima recibida por el usuario.
- **nota_min**: Un valor flotante que representa la calificación mínima recibida por el usuario.

- **valoraciones**: Un entero que representa el número de valoraciones recibidas por el usuario.

En la función push:

- **lista**: Un puntero a la estructura User_list donde se agregará el nuevo usuario.

En la función guardar_gustos:

- **lista**: Un puntero a la estructura User_list donde se buscará el usuario.
- **username**: Una cadena de caracteres que representa el nombre de usuario del usuario cuyos gustos se guardarán.
- **gustos**: Una cadena de caracteres que representa los gustos a guardar.
- **numero**: Un entero que representa la posición en la matriz likes donde se guardarán los gustos.

En la función limpiar_User_data:

- **guardar**: Un puntero a la estructura User_data que se liberará de la memoria.

En la función borrar_lista_de_usuarios:

- **lista**: Un puntero a la estructura User_list que se borrará, liberando la memoria asignada a los usuarios en la lista.

En la función initStack:

- **stack**: Un puntero a la estructura Stack que se inicializará.

En la función isStackEmpty:

- **stack**: Un puntero a la estructura Stack que se verificará si está vacía.

En la función pushRequest:

- **stack**: Un puntero a la estructura Stack donde se agregará la nueva solicitud de amistad.

- **sender**: Una cadena de caracteres que representa el remitente de la solicitud de amistad.
- **receiver**: Una cadena de caracteres que representa el destinatario de la solicitud de amistad.

En la función popRequest:

- **stack**: Un puntero a la estructura Stack de donde se eliminará la solicitud de amistad superior.

En la función enviarSolicitudAmistad:

- **stack**: Un puntero a la estructura Stack donde se almacenarán las solicitudes de amistad.
- **usuarioActual:** Una cadena de caracteres que representa el nombre de usuario del remitente de la solicitud de amistad.
- **usuarioDestino**: Una cadena de caracteres que representa el nombre de usuario del destinatario de la solicitud de amistad.

En la función valoracion:

- **user**: Un puntero a la estructura User_data del usuario al que se le dará una valoración.
- **nota_dada**: Un valor flotante que representa la valoración dada al usuario.

Las variables que encontramos más repetidas en las funciones del apartado “menu” son las siguientes:

- **other_user**: Es un puntero a una estructura User_data que representa a otro usuario
- **tl**: Es un puntero a una estructura timeline que representa una línea de tiempo.
- **flag**: Es una variable de tipo entero que se utiliza como una bandera para controlar los bucles while en varias partes del código.
- **submenu**: Es una variable de tipo entero que almacena la opción seleccionada en el menú secundario.

Finalmente, las variables que encontramos en el apartado “main” son las mismas que implementamos en los apartados anteriores.

2.1.2 ¿Cuáles fueron las estructuras de datos elegidas para este objetivo? ¿Cuál es su uso esperado?

Las estructuras de nuestro código están divididas en dos headers: publicación.h y usario.h

En primer lugar, analizaremos las de publicación.h:

```C/C++
typedef struct _post{
    char contenido[MAX_POST_LENGHT];
    User_data *usuario;
    struct _post* next;
    struct _post* prev;
}publicacion;

typedef struct{
    publicacion *first;
    publicacion *last;
    int size;
}timeline;

typedef struct _dicc{
    int counter;
    char *key;
    struct _dicc *next;
}diccionario ;
```

Estructura "_post":

La estructura "_post" representa una publicación en la red social. Tiene los siguientes campos:
"contenido": Es un array de caracteres (cadena de texto) que almacena el contenido de la publicación. El tamaño máximo de esta cadena está definido por la constante "MAX_POST_LENGHT".

"usuario": Es un puntero a una estructura "User_data" que contiene la información del usuario que realizó la publicación.
"next" y "prev": Son punteros a la siguiente y anterior publicación en una secuencia de publicaciones. Estos punteros permiten recorrer la lista de publicaciones en orden.

Estructura "timeline":

La estructura "timeline" representa la línea de tiempo de un usuario en la red social, que es una secuencia de publicaciones. Tiene los siguientes campos:
"first" y "last": Son punteros a la primera y última publicación en la línea de tiempo, respectivamente.
"size": Es un entero que indica la cantidad total de publicaciones en la línea de tiempo.

Estructura "_dicc":

La estructura "_dicc" representa un diccionario en el que se almacenan pares clave-valor. Tiene los siguientes campos:
"counter": Es un entero que lleva la cuenta del número de elementos en el diccionario.
"key": Es un puntero a una cadena de caracteres que representa la clave del elemento.
"next": Es un puntero a la siguiente entrada en el diccionario.
En resumen, estas estructuras se utilizan para modelar las publicaciones y la línea de tiempo en la red social, así como para implementar un diccionario que permite almacenar información asociada a claves. Cada una de estas estructuras tiene sus campos definidos para almacenar y relacionar los datos necesarios para el funcionamiento de la red social.

En segundo lugar, analizaremos las de usario.h:

```
C/C++
typedef struct _friend_request {
    char sender[MAX_USERNAME_LENGTH];
    char receiver[MAX_USERNAME_LENGTH];
    struct _friend_request* below;
} Friend_request;

typedef struct _stack {
    Friend_request* top;
} Stack;  hb
```

```
typedef struct _data{
    char username[MAX_USERNAME_LENGTH];
    char email[MAX_EMAIL_LENGTH];
    char password[MAX_PASSWORD_LENGTH];
    char city[MAX_CITY_NAME];
    int birth;
    int user_number;
    char likes[MAX_LIKES][MAX_LIKE_LENGTH];
    float nota;
    float nota_max;
    float nota_min;
    int num_valoraciones;
    char amigos[MAX_AMIGOS][MAX_USERNAME_LENGTH];
    Friend_request *solicitudes; //Esto hay que verlo
    struct _data* next;
    struct _data* prev;
}User_data;


typedef struct{
    User_data* first;
    User_data* last;
    int size;
}User_list;
```

Estructura "_friend_request":

La estructura "_friend_request" representa una solicitud de amistad enviada por un usuario. Tiene los siguientes campos:
"sender": Un array de caracteres que almacena el nombre de usuario del remitente de la solicitud.
"receiver": Un array de caracteres que almacena el nombre de usuario del destinatario de la solicitud.
"below": Es un puntero a la siguiente solicitud de amistad en una secuencia de solicitudes

Estructura "_stack":

La estructura "_stack" representa una pila de solicitudes de amistad. Tiene un único campo: "top": Es un puntero que apunta a la solicitud de amistad en la parte superior de la pila.

Estructura "_data":

La estructura "_data" contiene la información de un usuario en la red social. Tiene los siguientes campos:

- "username": Un array de caracteres que almacena el nombre de usuario del usuario.
- "email": Un array de caracteres que almacena la dirección de correo electrónico del usuario.
- "password": Un array de caracteres que almacena la contraseña del usuario.
- "city": Un array de caracteres que almacena el nombre de la ciudad en la que reside el usuario.
- "birth": Un entero que representa el año de nacimiento del usuario.
- "user_number": Un entero que identifica de manera única al usuario.
- "likes": Una matriz bidimensional de caracteres que almacena los "me gusta" (likes) dados por el usuario.
- "nota", "nota_max", "nota_min": Son valores de punto flotante que representan las calificaciones y estadísticas asociadas al usuario.
- "num_valoraciones": Un entero que indica el número de valoraciones recibidas por el usuario.
- "amigos": Una matriz bidimensional de caracteres que almacena los nombres de usuario de los amigos del usuario.
- "solicitudes": Es un puntero a la estructura "_friend_request" que representa las solicitudes de amistad recibidas por el usuario.
- "next" y "prev": Son punteros a la siguiente y anterior estructura "_data", respectivamente, lo que permite organizar los usuarios en una lista.

Estructura "User_list":

La estructura "User_list" representa una lista de usuarios en la red social. Tiene los siguientes campos:

- "first" y "last": Son punteros al primer y último usuario en la lista, respectivamente.
- "size": Un entero que indica la cantidad total de usuarios en la lista.

En resumen, estas estructuras se utilizan para almacenar y organizar la información de los usuarios, incluyendo sus solicitudes de amistad, en la red social. Cada estructura tiene sus campos definidos para almacenar los datos relevantes y permitir el acceso y manipulación de dichos datos.

2.1.3 ¿Cuál fue el algoritmo elegido para este objetivo?

El algoritmo escogido en este caso fue el Bubble Sort.

2.1.4 ¿Cuál es su comportamiento esperado?

El comportamiento esperado es ordenar la matriz de caracteres (char arr[]) en orden ascendente utilizando el algoritmo de ordenación de bubblesort.

Dado que utiliza la función strcmp para comparar las subcadenas de arr y la función strcpy para intercambiar los elementos, se espera que la matriz arr esté ordenada en orden ascendente al final de la ejecución. En nuestro caso lo utilizamos para ordenar las listas de usuarios.

2.1.5 ¿Qué hay de su desempeño en Big O?

El desempeño dado por esta función es O(m) pero normalmente nos encontramos que es O (n^2)

2.1.6 ¿Qué limitaciones tiene el algoritmo?

La limitación de este algoritmo es que cuando se tiene un gran número de elementos el tiempo requerido para ordenar el array aumenta significativamente. Dado que el algoritmo realiza múltiples comparaciones e intercambios, se vuelve menos eficiente a medida que va aumentando el tamaño del array. Resumidamente, tiende a ser más lento al tener arrays con muchos elementos.

2.1.7 ¿Qué se puede mejorar?

Que debería de estar implementado en una función y no en el código como tal

2.1.8 Tiempo necesario para desarrollar este objetivo

El tiempo estimado para desarrollar el objetivo descrito es de aproximadamente 4 horas.

2.1.9 Ubicación: a qué línea de código y de qué archivo pertenece esta implementación.
El primero se encuentra en menu.c y el header. Por otro lado el segundo Bubble short se encuentra en en publicacion.c y su respectivo header.

2.4 Objetivos obligatorios cumplidos

Durante el desarrollo del proyecto, hemos logrado cumplir todos los objetivos obligatorios establecidos en la asignatura.

1. En primer lugar, hemos implementado una Lista (List) y una Pila (Stack) funcionales como parte integral de diferentes funcionalidades del proyecto. La Lista ha sido utilizada en diversas ocasiones, por ejemplo, para listar los usuarios del sistema. La Pila ha sido implementada en el manejo de las solicitudes de amistad, ya que de esta forma el destinatario puede luego revisar las solicitudes pendientes.

2. Además, en el curso también hemos implementado de manera funcional uno de los algoritmos de búsqueda que se ha enseñado: el LinearSearch. Específicamente, hemos utilizado el algoritmo de búsqueda lineal (Linear Search) cada vez que ha sido necesario buscar elementos en las listas de usuarios dentro del programa.

1. Hemos implementado un algoritmo de ordenamiento aprendido en clase, en este caso hemos utilizado el Bubble Sort de manera funcional para ordenar la lista de usuarios por orden alfabético.

2. También, implementamos un Diccionario el cual guarda todas las palabras de las publicaciones en una estructura, a continuación esta se ordena mediante el algoritmo Bubble sort y las últimas palabras de este son las más repetidas, las cuales luego en tu perfil de usuario puedes consultar.

3. Cabe mencionar, que todas las partes importantes de nuestro código están detalladamente comentadas (Funciones, ciclos, secciones de código con una funcionalidad definida, secciones con una lógica difícil de entender a simple vista, entre otros).

4. Finalmente, recalcar que hemos usado github para que todos los miembros del equipo puedan trabajar a la par y más cómodamente.

En resumen, hemos logrado completar exitosamente todos los objetivos propuestos, lo que demuestra nuestro compromiso y habilidades en el desarrollo de este proyecto.

2.5 Objetivos deseables cumplidos

1. Conseguimos leer datos de una fuente externa como un archivo de texto o CSV en los cuales encontramos datos guardados como la lista de usuarios, las publicaciones y las peticiones de amistad.

2. Nuestro trabajo consiste en una red social con una temática distinta a la mayoría ya que se centra en el desarrollo de una red social única basada en un capítulo de la serie "Black Mirror". En esta plataforma, hemos decidido adoptar una temática con la visión distópica y provocativa de la serie.

   En esta red social en particular, hemos optado por eliminar otros tipos de interacciones tradicionales, como comentarios o likes, y en su lugar, nos basamos exclusivamente en el sistema de valoraciones de las publicaciones de los usuarios. Inspirados por el capítulo de la serie, buscamos explorar las implicaciones y consecuencias de una sociedad en la que la reputación y la aprobación social se basan únicamente en las valoraciones recibidas.

2.3 Objetivos exploratorios cumplidos

1. Medir el tiempo de ejecución para (como mínimo) tres funciones diferentes del código (Especialmente aquellas que reciban como parámetro de entrada una Lista o una estructura iterable similar).

2. La complejidad de tiempo de ejecución de tres funciones que contengan listas son los siguientes:

   La función “enviarSolicitudAmistad” tiene una complejidad de tiempo de O(n), donde n es el número de usuarios en la lista. Esto se debe a que la función recorre la lista de usuarios en un bucle while y realiza una comparación de cadenas utilizando strcmp en cada iteración. En el peor caso, si el usuario buscado se encuentra al final de la lista o no está presente, se recorrerán todos los usuarios en la lista, lo que lleva a una complejidad lineal O(n).

```
C/C++
void enviarSolicitudAmistad(User_list* lista, const char* nombre, const
char* otro) {
        User_data* usuarioActual = NULL;
        User_data* usuarioDestino = NULL;

        User_data* current = lista->first;
        while (current != NULL) {
            if (strcmp(nombre, current->username) == 0) {
                usuarioActual = current;
            }
            if (strcmp(otro, current->username) == 0) {
                usuarioDestino = current;
            }
            current = current->next;
        }

        if (usuarioActual != NULL && usuarioDestino != NULL) {
            FILE* file = fopen("/Users/naiara/Desktop/EDA2 3/amigos.txt",
"a");
            if (file == NULL) {
                printf("Error al abrir el archivo de solicitudes.\n");
                return;
            }

            fprintf(file, "%s %s\n", nombre, otro);
            fclose(file);
```

```
            printf("Solicitud de amistad enviada a %s y guardada en el
archivo.\n", otro);
        }
    }
```

La función “datosfichero” tiene una complejidad de tiempo O(n), donde n es el número de usuarios en la lista. Esto se debe a que recorre la lista de usuarios y escribe los datos de cada usuario en el archivo utilizando la función fprintf.

```
C/C++
int datosfichero(User_list* lista) {//Aquí lo que sí hay que pasar como
parametro es la estructura de la lista de usuario

    FILE* fp = fopen("/Users/senyo/CLionProjects/EDA2/Usuarios.txt", "w");
    if (fp == NULL) {
        printf("Error al abrir el archivo.\n");
        return NO_FILE_FOUND;
    }
    User_data *usuario = lista->first;
    while (usuario != NULL) {
        fprintf(fp, "%s %s %s %s %d %d %s %s %s %s %s %f %f %f %d\n",
usuario->username, usuario->email, usuario->password, usuario->city,
usuario->birth, usuario->user_number, usuario->likes[0],usuario->likes[1],
usuario->likes[2], usuario->likes[3], usuario->likes[4], usuario->nota,
usuario->nota_max, usuario->nota_min, usuario->num_valoraciones);
        usuario = usuario->next;
    }
    fclose(fp);
    return TRUE;
}
```

La función “postsfichero” tiene una complejidad de tiempo lineal O(n), donde 'n' es el número de publicaciones en el timeline. Esto se debe a que recorre todas las publicaciones una vez para escribirlas en el archivo. El tiempo de ejecución aumenta proporcionalmente al número de publicaciones en el timeline.

```C/C++
int postsfichero(timeline *tl){
    FILE *fp = fopen("/Users/naiara/CLionProjects/EDA2/posts.txt", "w");
    if (fp == NULL){
        printf("Error al abrir el archivo.\n");
        return NO_FILE_FOUND;
    }
    publicacion *post = tl->first;
    while (post != NULL){
        fprintf(fp, "%s\n", post->contenido);
        fprintf(fp, "%s\n", post->username);
        post = post->next;
    }
    fclose(fp);
    return TRUE;
}
```

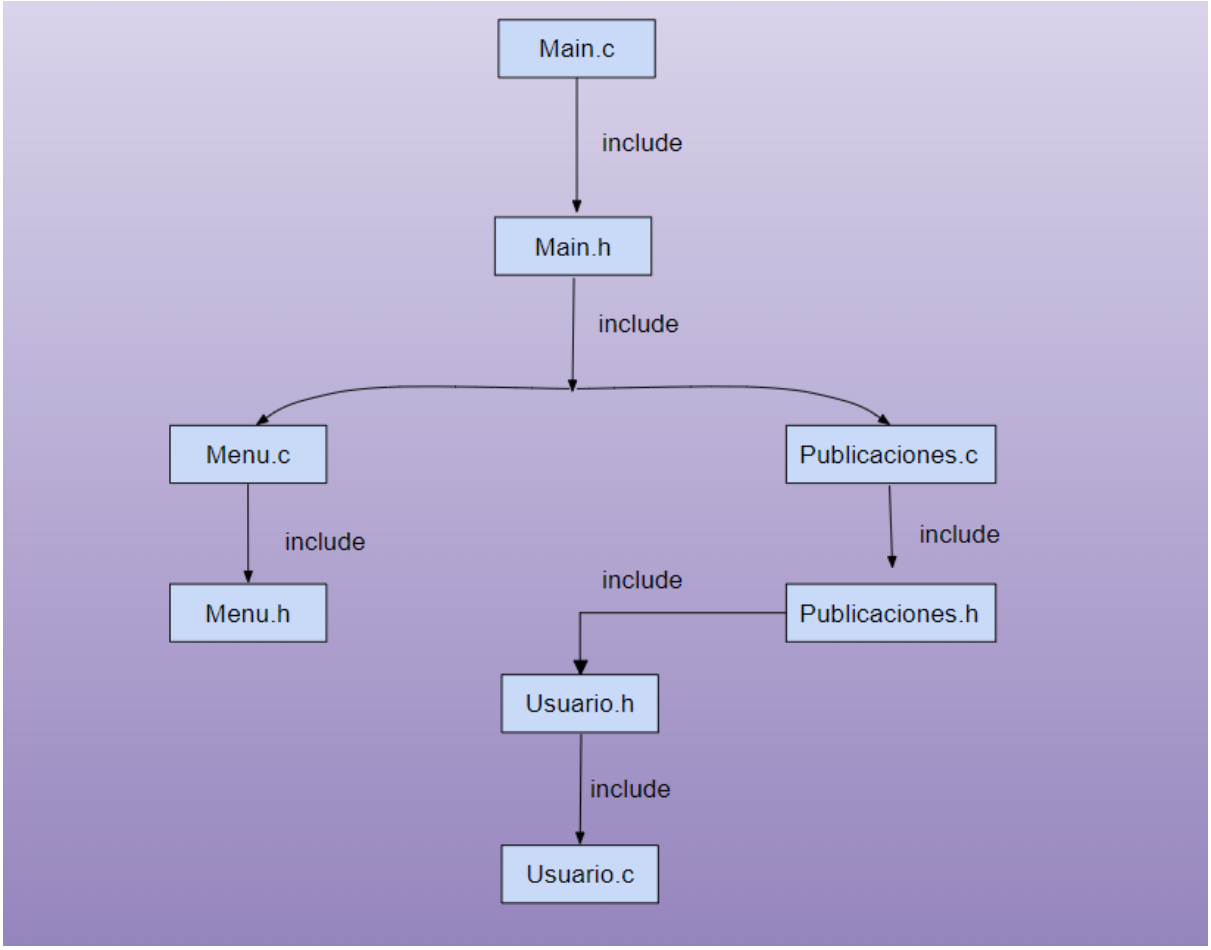
3. Conseguimos Implementar las funcionalidades de lectura y guardado de datos (Información de los usuarios y sus contactos, y demás que se quiera añadir) desde un archivo externo (.txt o csv)

3. SOLUCIÓN

3.1 Arquitectura del sistema

La forma en la que hemos decidido organizar nuestro trabajo es por una parte en Main.h, Menu.h, Publicacion.h y Usuario.h en los cuales están las funciones principales y por otro lado está el Main.c, Menu.c, Publicacion.c y Usuario.c donde se implementan las funciones anteriores.

Arquitectura de EDAHANA



Mainc.c: Sirve como el punto de partida desde el cual se comienza a ejecutar el programa y también puede recibir argumentos de línea de comandos. En este se ejecutan todas las funciones del programa.

Menú.c: Sirve para acceder a los comandos y a las diferentes ventanas de la red social.

Publicacion.c: Contiene la implementación de varias funciones relacionadas con la manipulación de las publicaciones que enviará cada usuario.

Usuario.c: Se implementan funciones relacionadas con la información de cada perfil y las solicitudes de amistad.

Después tenemos los headers o los puntos .txt :

En resumen, los archivos .h, como "menu.h", son importantes para organizar y modularizar el código de un programa, separando las declaraciones de las implementaciones. Además, permiten una mejor reutilización de código al proporcionar interfaces claras y consistentes para acceder a funciones y estructuras definidas en otros archivos.

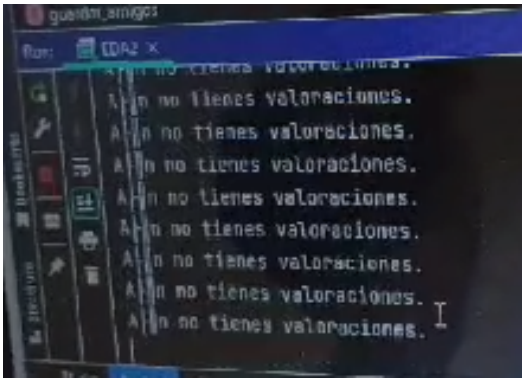
3.2 Gestión de errores

Durante nuestro proceso para crear la red social tuvimos multitud de errores diferentes pero los más significativos fueron los siguientes:

La función “create_user” presentó un error debido a que contenía otra función interna llamada “cambiar_ciudad”, la cual asignaba el nombre de la ciudad al usuario en sus datos. Sin embargo, esta función generaba un error. Para solucionarlo, decidimos eliminar la función “cambiar_ciudad” y en su lugar incorporamos directamente el código dentro de la función create_user. De esta manera, todos los datos necesarios fueron correctamente capturados por create_user.

En el programa desarrollado para nuestra red social en C, hemos identificado un error crítico que afecta al funcionamiento correcto del código. Este error se relaciona con la falta de cierre adecuado de los bucles "while", lo que resulta en ciclos infinitos y un comportamiento incorrecto del programa.

El problema radica en que hemos olvidado proporcionar una condición que permita salir del bucle, es decir, no hemos establecido una condición que se evalúe como falsa en algún momento. Como resultado, los bucles "while" se ejecutan infinitamente, causando que el programa se quede



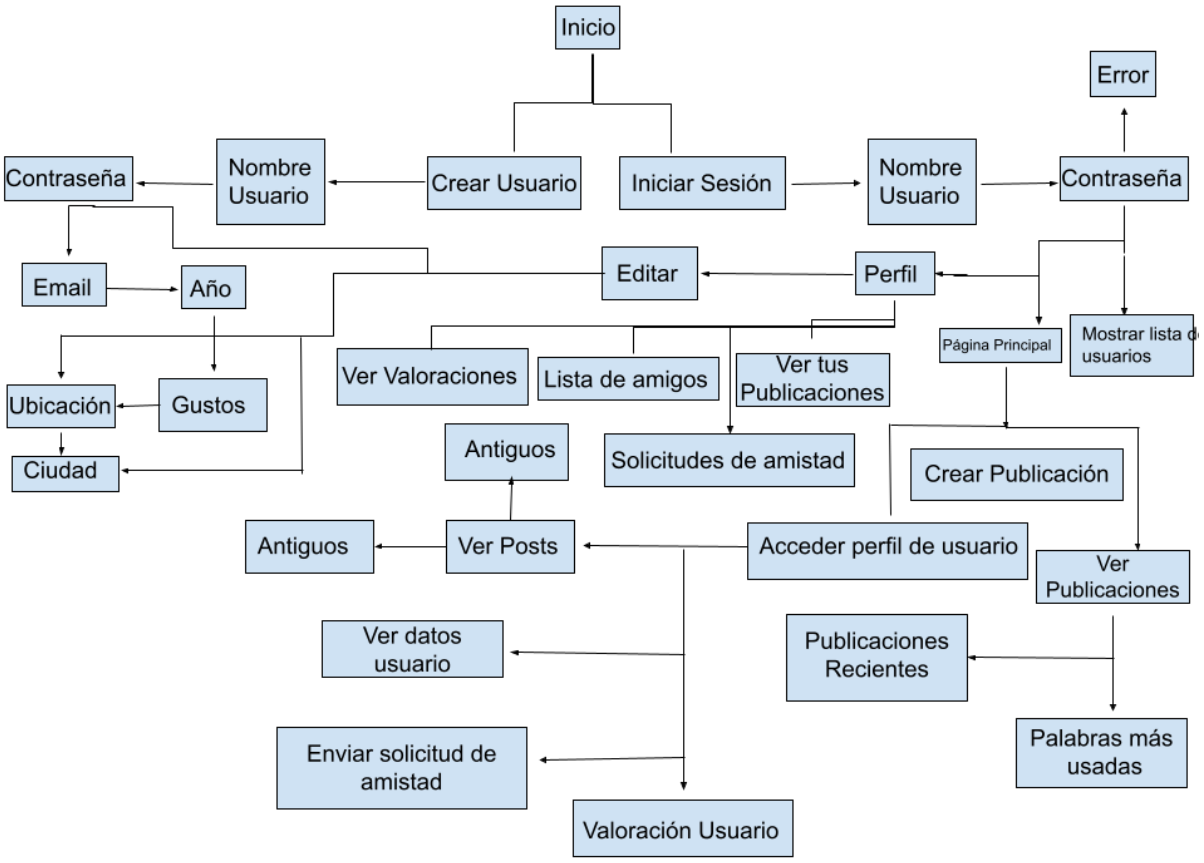
atrapado en un bucle infinito y produciendo una salida continua de información.

Para solucionar este error, es fundamental revisar cuidadosamente todos los bucles "while" en el código y asegurarse de proporcionar una condición apropiada que se evalúe como falsa en algún momento.

Para acabar, hemos identificado una serie de errores que a día de hoy no hemos conseguido resolver. El principal ejemplo es el diccionario, ya que a simple vista debería estar correcto, sin embargo, la función “contar palabras” no se ejecuta correctamente y por lo tanto, da error.

Por último, para que el programa estuviera perfectamente programado se debería hacer un análisis de errores más exhaustivo ya que de esta manera si el usuario introduce una respuesta incorrecta a la que el programa está pidiendo este no petaría.

3.3 Diseño de modelo de datos



Al iniciar nuestra red social, los usuarios tienen dos opciones disponibles: iniciar sesión o crear un nuevo usuario. En el caso de optar por la creación de un nuevo usuario, se solicitará la siguiente información: nombre de usuario, contraseña, correo electrónico, año de nacimiento (en el rango de 1920-2023), cinco gustos, ubicación y ciudad. En el caso de seleccionar la opción de "iniciar sesión", se realizará una verificación para asegurar que el usuario y la contraseña coincidan. Si no coinciden, se mostrará un mensaje de error. Una vez iniciada la sesión, los usuarios tendrán acceso a un menú con tres opciones: "Mostrar lista de usuarios", "Perfil" y "Página principal". Si seleccionan la opción "Perfil", podrán editar su información personal y ver los datos generales de tu usuario (valoraciones, publicaciones y amigos). Si optan por la opción "Página principal", podrán crear nuevas publicaciones, ver las publicaciones existentes y explorar las palabras más utilizadas en ellas. Además, tendrán la posibilidad de acceder a los perfiles de otros usuarios, donde podrán visualizar toda la información disponible sobre ellos.

3.4 Descripción y procesamiento del conjunto de datos

Para guardar los datos del programa, utilizamos principalmente archivos CSV y archivos de texto. Estos archivos nos permiten almacenar la información de la lista de usuarios, las publicaciones y las solicitudes de amistad.

En estos archivos cada línea representa una fila de datos, y los valores de cada columna están separados por comas. Esto facilita la lectura y escritura de los datos en forma de tabla.

REFERENCIAS

- OpenAI. (2021). OpenAI GPT. Retrieved from https://openai.com/research/gpt

- Creating a Queue in C | DigitalOcean. https://www.digitalocean.com/community/tutorials/queue-in-c

- parzibyte. «Separar cadena a partir de delimitadores en C con strtok». Parzibyte's blog, 13 de noviembre de 2018. https://parzibyte.me/blog/2018/11/13/separar-cadena-delimitadores-c-strtok

- 262588213843476. «A Key/Value Dictionary System in C». Gist, https://gist.github.com/kylef/86784

- C Program to Sort Names in Alphabetical Order. https://www.tutorialspoint.com/c-program-to-sort-names-in-alphabetical-order

- C program to read string with spaces using scanf() function. C program to read string with spaces using scanf() function (includehelp.com)

- maro. «chcp 65001 and a .bat file». Stack Overflow, 23 de mayo de 2017 command line - chcp 65001 and a .bat file - Stack Overflow

- 262588213843476. «Poner Acentos y La Tilde de La ñ En C». Gist https://gist.github.com/andreandyp/341274d5597e683559ae9d1b4aa661aa

- Joe wright (Director) Laurie Borg Charlie Brooker Ian Hogan Annabel Jones Angela Philips (Productores) (2011–2019) Nosevide (Temporada 3 Episodio 1)*Black Mirror*[Serie] Zeppotron, House of Tomorrow.