Universitat Pompeu Fabra Barcelona

DATA STRUCTURES AND ALGORITHMS II, 2022-2023

PROJECT TITLE

Enric Vives Arnau \\ U199773

PLANTILLA DE INFORME DE PROYECTO

TABLA DE CONTENIDOS

---

1. INTRODUCCIÓN

El propósito de este informe de proyecto es documentar el desarrollo e implementación de un proyecto de código específico. Esta sección proporciona una visión general del contexto, la definición del problema, las soluciones existentes revisadas para obtener inspiración, la propuesta, los objetivos alcanzados y la estructura del informe.

El contexto de este proyecto es la creación de una red social temática (app para conocer gente), mediante la implementación de listas enlazadas y diferentes algoritmos vistos en clase y no, aparte de la implementación de otras estructuras, como podrían ser diccionarios. La declaración del problema tuvo como objetivo abordar la forma con la que ibas tanto a trabajar y qué tipo de código usamos, es decir, en este caso si llevar a cabo la mayor parte del código mediante tablas hash o si hacerlo con listas enlazadas, finalmente la opción elegida fue la segunda. Durante la fase de planificación, se estudiaron diversas soluciones existentes para obtener inspiración y comprender los enfoques más adecuados para resolver el problema en cuestión. Basándose en esta investigación, se formuló una propuesta que describe los objetivos a alcanzar mediante la implementación del código.

Los objetivos alcanzados por el proyecto se discutirán en detalle en la sección siguiente. El informe está estructurado de la siguiente manera: la Sección 2 abarca los objetivos del proyecto, la Sección 3 profundiza en la solución desarrollada y la Sección 4 proporciona una descripción exhaustiva del conjunto de datos utilizado y su procesamiento.

2. OBJETIVOS DEL PROYECTO

Esta sección describe los objetivos que se lograron mediante la implementación del proyecto de código. Cada objetivo se presenta en las subsecciones siguientes, que proporcionan una visión general de su implementación, las estructuras de datos y algoritmos utilizados, su comportamiento esperado y limitaciones. También se indica la ubicación de cada implementación dentro del código fuente.
Aclarar, que aparte de la estructura dada por el profesorado basada en objetivos obligatorios, deseados y voluntarios, se ha seguido en todo momento teniendo en cuenta estos la guía de buenas prácticas, la cual si tenemos en cuenta está como objetivos de la práctica, se han cumplido muchas más funcionalidades que no se explican en el siguiente fragmento de documento pero que en el video explicación sí que tendrán su mención.

2.1. Objetivos Obligatorios Cumplidos

Los objetivos obligatorios cumplidos por el proyecto son los siguientes:

Objetivo 1:

Este objetivo se basaba en implementar Listas(List), Pilas(Stack) y Colas(Queues) como parte de algunas funcionalidades del proyecto. Nuestra forma de implementación fue la siguiente:

List.** Las funciones de nuestro proyecto, como he comentado anteriormente en la introducción, se basan mayormente en listas enlazadas para poder llevar a cabo todas las funcionalidades requeridas para el funcionamiento del mismo. Esto se puede ver en cualquiera de las estructuras en el fichero utils.h donde están todas estas declaraciones.

Pero lista como tal se puede comprobar como sé pedía en el guardado durante la realización de publicaciones por los usuarios.

Stack.** Las pilas en nuestro proyecto están implementadas como se sugería en la guía de buenas prácticas proporcionadas por el profesorado, se implementó en la función de agregar amigos desconocidos, en nuestro proyecto consistía en dar likes a diferentes usuarios aleatorios de los cuales con una funcionalidad exploratoria que se explicará más adelante se les podía ver la información de este.

Queue.** Las colas como el Stack estaba sugerido, y así se implementó en la gestión de solicitudes de amistad, que en nuestro caso, pasada nuestra capa estética consistía en la gestión de los likes que te había dado otro usuario. Esta gestión consiste en que presionando 1 se acepta la solicitud y se guarda el amigo en una cola para más adelante poder imprimir mis amigos/matches y si se presiona 0 se deniega y esa solicitud se borra del sistema.

Ubicación en el código: Utils.c & Utils.h

****Objetivo 2:****

Este objetivo está basado en implementar uno de los algoritmos de búsqueda que hemos visto a clase, concretamente el algoritmo que elegimos fue el Liear Search, este se implementó para una funcionalidad extra de la aplicación, la cual es que una vez registrado dentro del submenú, no podías ver la información de los usuarios, ya que la funcionalidad de imprimir la lista, estaba en el menú principal, así que mediante el algoritmo de búsqueda de LinearSearch lo logramos.

Esta implementación funciona de la siguiente manera:

El código se planteó con dos funciones diferentes, LinearSearch y buscar_usuario_por_nombre. El algoritmo linear_search implementa una búsqueda lineal en un arreglo de caracteres (arr) para encontrar una cadena objetivo (target). La función retorna un entero que indica el resultado de la búsqueda. La función buscar_usuario_por_nombre utiliza el algoritmo linear_search para buscar un nombre de usuario ingresado por el usuario en una lista enlazada de usuarios representada por el puntero userList. En resumen, el algoritmo linear_search realiza una búsqueda secuencial en un arreglo de caracteres, mientras que la función buscar_usuario_por_nombre utiliza este algoritmo para buscar un nombre de usuario en una lista enlazada de usuarios.

Ubicación en el código: Utils.c

****Objetivo 3:****

Este objetivo consistía en Implementar funcionalmente uno de los algoritmos de ordenamiento que se han visto a lo largo del curso (InsertionSort, BubbleSort, MergeSort o QuickSort). El algoritmo elegido fue el InsertionSort, después de buscar la idea de donde poder implementarlo dentro del código obligatorio, decidimos crear una función extra, la cual aporta valor al motivo de nuestra red social, ya que aplicamos el InsertionSort para ordenar

los usuarios por edad, y creímos que era interesante, ya que daba una visión o una especie de filtro manual de edad, que en una red social para conocer gente aporta mucho valor.

El algoritmo funciona de la siguiente manera:

El algoritmo de ordenación por inserción utilizado en la función ordenar_por_edad recorre la lista original de usuarios y, en cada iteración, extrae un nodo de la lista original y lo inserta en la posición correcta dentro de una nueva lista ordenada. Al final, se imprime la lista ordenada y se libera la memoria asignada a los nodos. De esta manera, se logra ordenar la lista original por la edad de los usuarios.

Ubicación en el código: Utils.c

****Objetivo 4:****

Siguiendo con el cuarto objetivo obligatorio, se tenía que Implementar o bien un Diccionario (Dictionary), o bien una tabla Hash (HashTable) funcionales. Nosotros decidimos implementar un diccionario por dos motivos principales: primeramente su facilidad de uso según nuestro punto de vista y después la flexibilidad de claves delante de las tablas hash.

Esto se aplicó como se sugería, haciendo una especie de contador de palabras que contaba la aparición dentro de las publicaciones de los diferentes usuarios, para más tarde poder hacer un top 10 de las palabras en tendencia, nuestra forma de aplicarlo fue hacer que las palabras en tendencia fueran para cada usuario en concreto es decir no un top 10 palabras globales en tendencia sino un top10 diferente para cada uno. La idea de esto sería para si hubiera una programación más a fondo posteriormente para poner en el mercado la app poder crear un sugeridor de palabras que más usa ese usuario.

En términos generales, se utiliza un diccionario para contar la frecuencia de las palabras en un conjunto de publicaciones. Recorre cada publicación y divide su contenido en palabras individuales. Luego, mantiene un diccionario donde cada palabra se asocia con la cantidad de veces que aparece en las publicaciones. Si la palabra ya está en el diccionario, se incrementa su contador; de lo contrario, se agrega al diccionario con un contador inicial de 1. Al final, se ordena el diccionario por frecuencia de aparición y se imprimen las palabras más usadas. Finalmente, se libera la memoria utilizada por el diccionario.

Ubicación en el código: Utils.c & Utils.h

****Objetivo 5:****

Este objetivo consiste en que todas las partes importantes de tu código deben estar comentadas (Funciones, ciclos, secciones de código con una funcionalidad definida, secciones con una lógica difícil de entender a simple vista, entre otros). Este objetivo, dentro de nuestro entender, está cumplido, ya que como sé pedía todas las partes esenciales del código han sido comentadas siguiendo la guía proporcionada por el profesorado.

****Objetivo 6:****

Para finalizar, el último objetivo obligatorio era usar el github para que todos los miembros del equipo puedan trabajar a la par y más cómodamente. Y que los diferentes miembros del equipo deben mostrar actualizaciones del código constantes.

Este objetivo también ha sido cumplido con éxito, ya que en el comit del git hub se pueden ver que se ha trabajado actualizando de forma constante el código haciendo los push y pulls correspondientes.

2.2. Objetivos Deseables Cumplidos

Los objetivos deseables cumplidos por el proyecto son los siguientes:

****Objetivo 1:****

Se pedía que la Red Social fuera temática y tuviera una capa de personalización a nivel conceptual que la haga más especial entre los demás proyectos.

Así que de esta forma nos surgió la idea de crear nuestra red social 'Meet Me', una red social para conocer gente y poder entablar una conexión mediante el correo electrónico, ya tienes la posibilidad de lanzar un like a otro usuario después de haber visto su información o de forma desconocida para hacerlo más interesante, y una vez el otro usuario te acepta la solicitud de like se hace match, y en ese momento si quieres puedes chequear su información para ver sus gustos o localidad y mediante el correo electrónico poder entablar una relación de amistad o no, fuera de nuestra aplicación.

Así que creemos que con esta personalización de la red social a nivel conceptual, cumplimos con éxito los requisitos del objetivo.

Ubicación en el código: Main.c, Utils.c & Utils.h

****Objetivo 2:****

El siguiente objetivo deseable era que consiguiéramos 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).

Así que tal y como se pedía en el objetivo, elegimos tres funciones que reciben como parámetro de entrada una lista y les medimos el tiempo de ejecución. Estas tres funciones concretamente son:

Imprimir_solicitudes_aceptadas
Imprimir_solicitudes_amistad
agregar_amigos_random

El cálculo del tiempo de ejecución de estas funciones procede de la misma forma:

Registra el tiempo de inicio y finalización de la ejecución del código y calcula la diferencia para obtener el tiempo de ejecución total. Luego, muestra este tiempo en segundos utilizando la función printf(). Esto proporciona una indicación del tiempo que tarda en ejecutarse la función.

Ubicación en el código: Utils.c & Utils.h

2.3. Objetivos Extras Cumplidos

*Aquí creemos importante volver a aclarar que no incluirá como objetivos extras todas las recomendaciones de la guía de buenas prácticas, como objetivos extras, pero es importante recalcar que se han cumplido cada una de las recomendaciones.

Los objetivos extras cumplidos por el proyecto son los siguientes:

****Objetivo 1:****

En los requisitos, decía que se implementara en el código la opción de leer y cargar la información de un archivo al ejecutarse el programa para de esta manera no tener que introducir usuarios manualmente, y que sino se quería hacer quemáramos código (hard-coding) para introducir directamente 10 usuarios. Nuestra decisión final fue hacerlo de esta última forma.

Esta decisión fue tomada así, ya que le queríamos dar un poco más de aleatoriedad a nuestra red social cada vez que fuera ejecutado el programa, así que creamos una función la cual genera 10 usuarios con todo su typedef, es decir, sus alias, edades, correo electrónico… asignado de forma aleatoria.

Este código funciona de la siguiente manera:

La función está dividida en dos más pequeñas para que así el menú principal pueda llamar a la segunda desde otra clase. Es decir, en la clase main se encuentra la función generar_usuario_proceso la cual solo llama a la segunda llamada generar_usuarios, que se encuentra en la clase utils.c y esta contiene todo el código necesario para el funcionamiento de esta.

El algoritmo utilizado en el código proporcionado se conoce como algoritmo de mezcla aleatoria (random shuffle algorithm). Este algoritmo utiliza nombres, gustos y ciudades predefinidos y los combina de forma aleatoria para generar usuarios con datos aleatorios, como nombres de usuario, correos electrónicos, gustos, ubicaciones, edades y alias. Estos usuarios se agregan a una lista enlazada de usuarios representada por el parámetro userList.

****Objetivo 2:****

Hay otro objetivo extra que cree para darle más valor a la red social y sobre todo también para que la estética de esta tuviera aún más fuerza. Esta consiste en una función la cual se llama desde el submenú una vez ya has iniciado sesión en la red social, que imprime las solicitudes que este usuario tiene pendientes para aceptar.

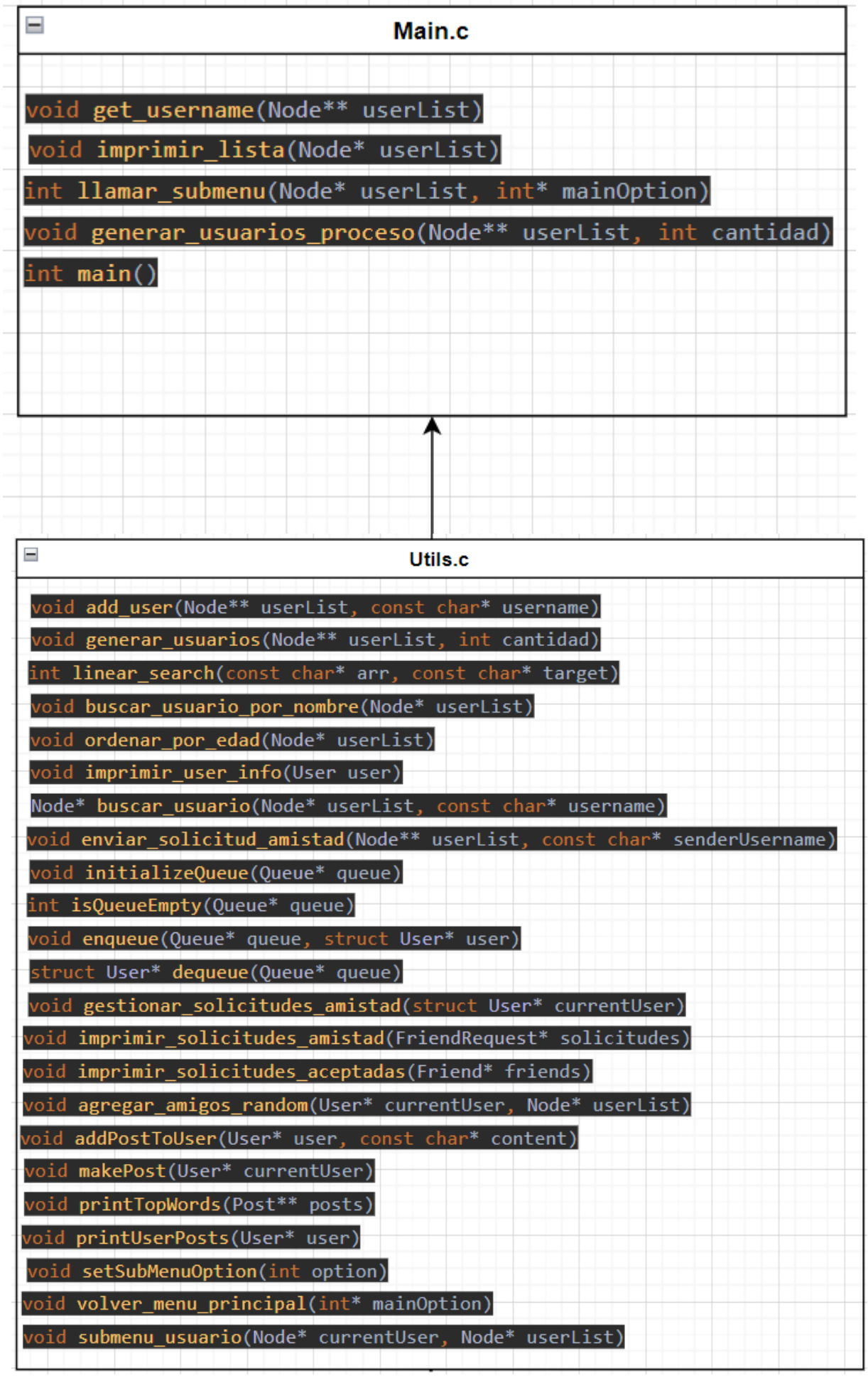
Esta funcionalidad está inspirada en redes sociales como podría ser Instagram donde tienes una lista de solicitudes pendientes de aceptar. Salió la necesidad de crear esta función, ya que había un punto oscuro, el cual era que en los requisitos pedían que se pudiera mostrar las solicitudes aceptadas, en nuestro caso los matches, pero claro para gestionar las solicitudes y poder ampliar la lista de solicitudes de amistad, se tenía que hacer de forma mental, recordarse de todos los nombres de usuario que te habían dado like, entonces implementamos esta función que al llamarla te muestra por la pantalla de la consola las solicitudes que tienes pendientes.

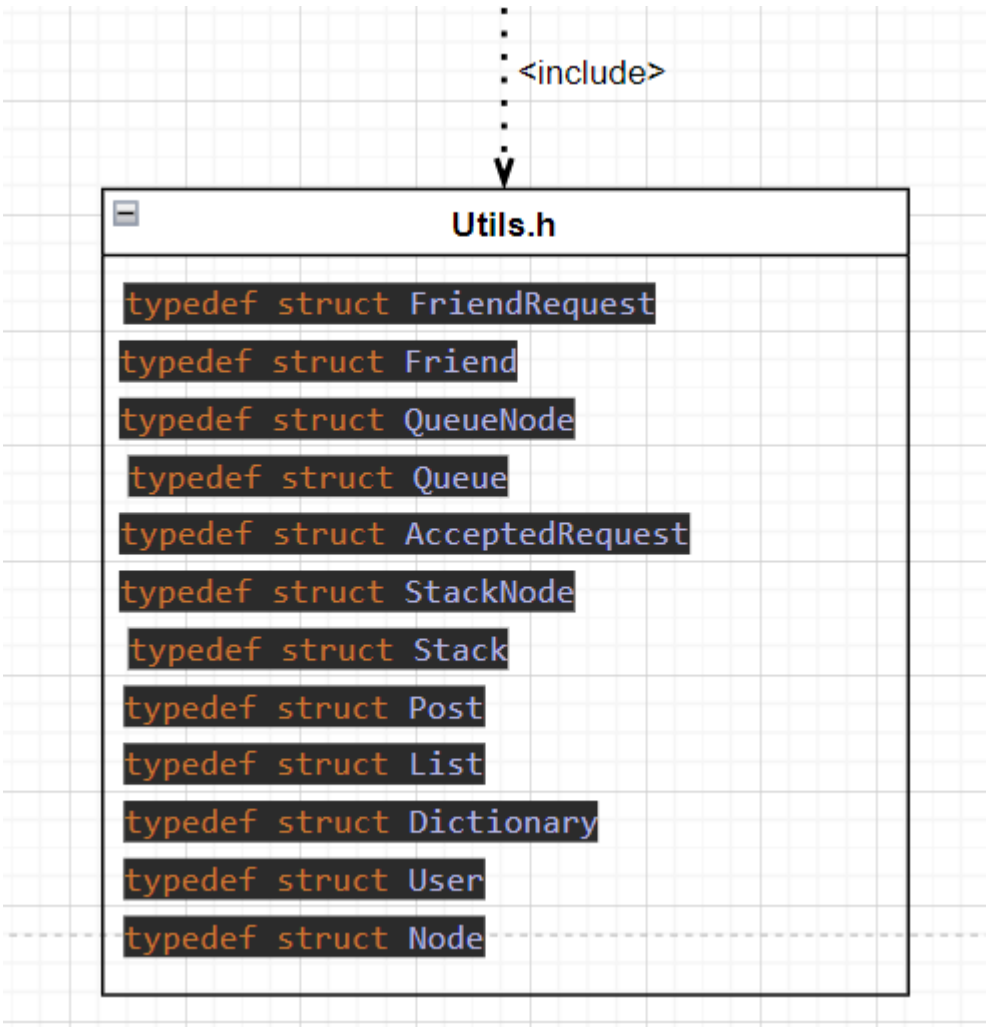
3. SOLUCIÓN

Esta sección profundiza en la solución desarrollada para el proyecto, proporcionando una visión general de la arquitectura del sistema, los mecanismos de manejo de errores, el diseño del modelo de datos y la descripción y el procesamiento del conjunto de datos.

3.1. Arquitectura del Sistema

La arquitectura del sistema es un aspecto crucial del proyecto. Determina la organización y la interacción de varios componentes para lograr la funcionalidad deseada. El siguiente diagrama ilustra la arquitectura del sistema, resaltando los principales bloques y sus funcionalidades respectivas.


Main.c
void get_username(Node** userList)
void imprimir_lista(Node* userList)
int llamar_submenu(Node* userList, int* mainOption)
void generar_usuarios_proceso(Node** userList, int cantidad)
int main()
Utils.c
void add_user(Node** userList, const char* username)
void generar_usuarios(Node** userList, int cantidad)
int linear_search(const char* arr, const char* target)
void buscar_usuario_por_nombre(Node* userList)
void ordenar_por_edad(Node* userList)
void imprimir_user_info(User user)
Node* buscar_usuario(Node* userList, const char* username)
void enviar_solicitud_amistad(Node** userList, const char* senderUsername)
void initializeQueue(Queue* queue)
int isQueueEmpty(Queue* queue)
void enqueue(Queue* queue, struct User* user)
struct User* dequeue(Queue* queue)
void gestionar_solicitudes_amistad(struct User* currentUser)
void imprimir_solicitudes_amistad(FriendRequest* solicitudes)
void imprimir_solicitudes_aceptadas(Friend* friends)
void agregar_amigos_random(User* currentUser, Node* userList)
void addPostToUser(User* user, const char* content)
void makePost(User* currentUser)
void printTopWords(Post** posts)
void printUserPosts(User* user)
void setSubMenuOption(int option)
void volver_menu_principal(int* mainOption)
void submenu_usuario(Node* currentUser, Node* userList)



La arquitectura del sistema consta de varios componentes clave, que incluyen las tres clases usadas en el proyecto con sus respectivas funciones dentro de cada clase. Estas clases son el Main.c el cual tiene el menú principal y llama a funciones de la segunda clase utils.c, y esta tiene el submenú y el resto de funcionalidades las cuales incluyen a la última clase la e utils.h, esta tiene todas las estructuras que usa el sistema.

3.2. Manejo de Errores

El manejo efectivo de errores es esencial para mantener la robustez y confiabilidad del proyecto de código. Esta subsección analiza cómo se manejaron los errores en el proyecto y la justificación detrás de los mecanismos de manejo de errores elegidos.

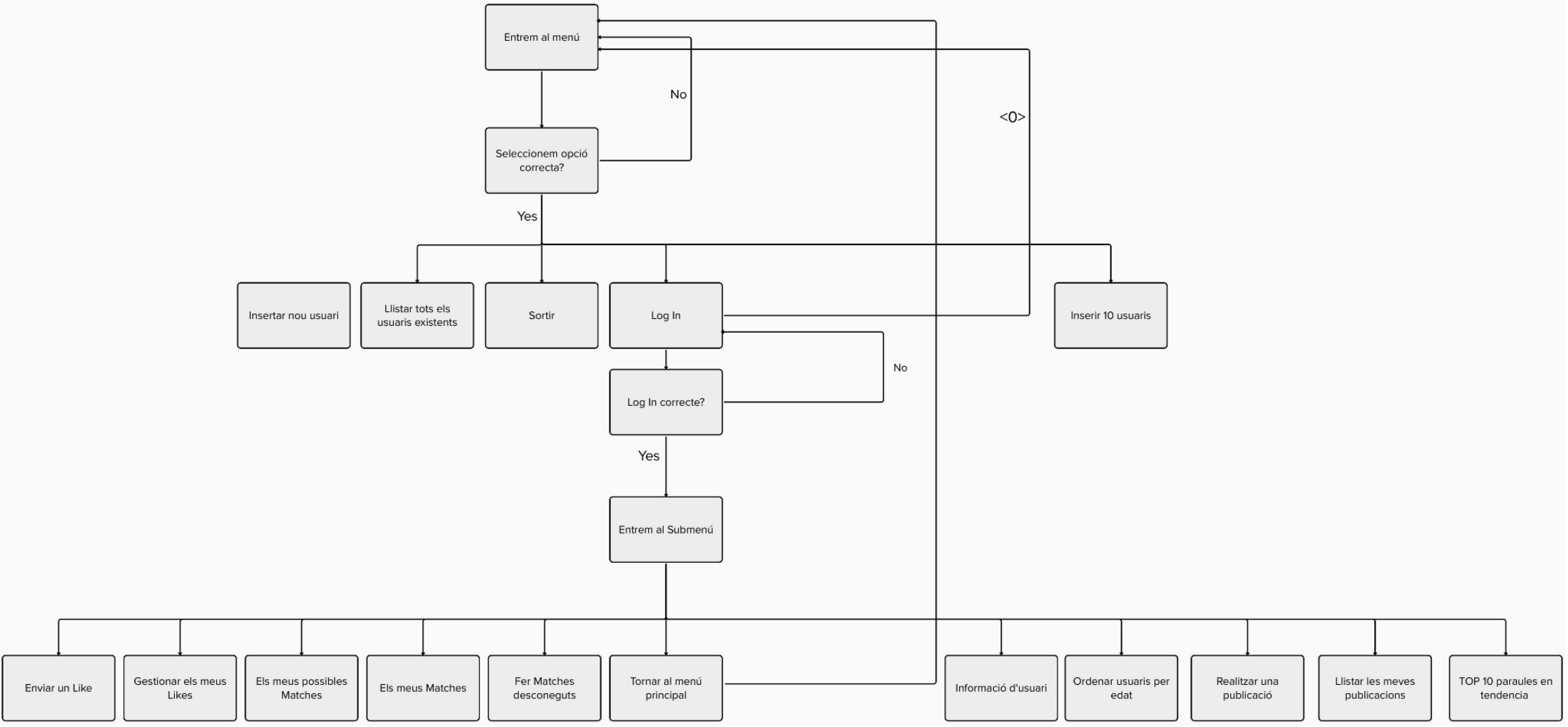
La estrategia que llevamos a cabo en nuestro proyecto fue la siguiente, para poder controlar cualquier entrada de datos inesperada por parte del cliente que hiciera que una vez compilado el programa fallase, entrase en un bucle eterno o cualquier otro tipo de error, empleamos una frontera para cada vez que el usuario tenía que introducir algo. Esta frontera lo que controla es que si un usuario no introduce exactamente lo que queremos que notifique que la entrada ha sido errónea y pida otra vez que se introduzcan los datos.

Luego hay la parte de arreglar los errores en el código antes de poder compilarlo, que para poder hacer esta tarea, nos ayudó muchísimos la herramienta predeterminada de sugerencia para solucionar errores del Clion, ya que algunas veces no, pero la mayoría de estas nos conducía por el buen camino a la hora de arreglar esa línea de código o si el error estaba en otra parte del código, nos conducía hacia allí.

Si todo esto no funcionaba, y no encontrábamos donde estaba el error, al ser un puntero escondido o cualquier cosa, usábamos la herramienta de debugging para descubrir dónde se encontraba el error y poder solucionarlo.

3.3. Diseño del Modelo de Datos

El diseño del modelo de datos juega un papel crucial en garantizar el flujo y la gestión adecuados de los datos dentro del proyecto. Esta subsección presenta un diagrama de flujo de datos que visualiza el flujo de datos a través de la solución, junto con una descripción de los principales componentes y sus interacciones.


Entrem al menú
No
<0>
Seleccionem opció correcta?
Yes
Insertar nou usuari
Llistar tots els usuaris existents
Sortir
Log In
Inserir 10 usuaris
No
Log In correcte?
Yes
Entrem al Submenú
Enviar un Like
Gestionar els meus Likes
Els meus possibles Matches
Els meus Matches
Fer Matches desconeguts
Tornar al menú principal
Informació d'usuari
Ordenar usuaris per edat
Realitzar una publicació
Llistar les meves publicacions
TOP 10 paraules en tendencia

El diseño del modelo de datos comprende el proceso que sucede cuando ejecutamos nuestro código que hemos plasmado en este diagrama de flujo, si seguimos el diagrama en orden descendente, la primera acción que nos sale és la de entrar al menú, seguidamente cuando ya vemos el menú hemos de seleccionar una de las cinco opciones posibles, si no seleccionamos una de estas volvemos otra vez al menú principal pero si las seleccionamos cada una hará su acción en cambio la opción de Log In (sí la hacemos correctamente) nos conducirá hacia un submenú donde veremos 11 opciones diferentes las cuales son las siguientes:
Enviar un like, gestionar mis likes, mis posibles matches, mis matches, hacer matches desconocidos, volver al menú principal, información de usuario, ordenar usuarios por edad, realizar una publicación, para que te conozcan un poco más, listar mis publicaciones, TOP 10 palabras en tendencia

3.4. Descripción y Procesamiento del Conjunto de Datos

El conjunto de datos utilizado en el proyecto influye significativamente en el resultado y rendimiento de la solución implementada. Esta subsección proporciona una descripción de los conjuntos de datos utilizados, junto con una explicación de cómo se leyeron y procesaron.

Los conjuntos de datos, en este proyecto optamos para que no fueran externos, es decir como un archivo CSV o un archivo JSON, si no que todos y cada uno de los datos fueran introducidos por el usuario, ya que siendo una red social con una temática tan personal creímos oportuno que el usuario pudiera tomar todas y cada una de sus decisiones.

Los conjuntos de datos se procesaron mediante la función scanf para leer datos de entrada desde el usuario, aunque alguna vez también se usó fgets pero solo en momentos muy puntuales. Esta función permite obtener datos de diferentes tipos, como números enteros, números de punto flotante y cadenas de caracteres. Se especifican los formatos de entrada adecuados en función de los tipos de datos esperados. Por ejemplo, se obtuvieron %d para leer números enteros y %f para leer números de punto flotante.

Todos estos datos fueron tratados como hemos explicado anteriormente para la prevención de errores, es decir, que solo pasaran el filtro los correspondientes. Los datos leídos mediante escaneo se almacenan en variables adecuadas para su posterior procesamiento.

##4. REFERENCIAS:

1- Microsoft. (s.f.). Comentarios en C. Recuperado de:
https://learn.microsoft.com/es-es/cpp/c-language/c-comments?view=msvc-170

2- Corrector Castellano. (s.f.). Corrector Ortográfico en Línea. Recuperado de:
https://www.corrector-castellano.com/

3- YouTube. (2018). Cómo utilizar diccionarios en lenguaje C. [Video]. Recuperado de: https://www.youtube.com/watch?v=qjsjs8nk-kI

4- YouTube. (2020). Listas enlazadas doble puntero en lenguaje C. [Video]. Recuperado de: https://www.youtube.com/watch?v=QH64qSW7wM8

5- Toda la información de teoría proporcionada por la universidad, no solo de este año si no toda la encontrada,

** Entre muchas otras páginas web, que se utilizaron de forma puntual para solucionar diferentes errores y dudas.