social-web-simulator

Generated by Doxygen 1.13.0

Chapter 1

Simulador de Red Social

Este proyecto simula una **Red Social**, utilizando estructuras de datos complejas como los grafos, listas enlazadas y tablas hash, aparte de algoritmos vistos anteriormente como quicksort, Jaccard y Dijkstra.

1.1 Resumen

El simulador se puede resumir de la siguiente manera:

- Creación de usuarios aleatorios.
- Calculo del índice de Jaccard para comparar la similitud entre los usuarios.
- Creación de las conexiones entre los nodos.
- Simulación de las publicaciones.
- Creación de una imagen gráfica del grafo.

1.2 Uso del Programa

Antes de ejecutar el programa, es necesario compilarlo de manera adecuada y seguir los pasos recomendados a continuación:

1.2.1 1. Recomendación Inicial

Apenas posea el programa en su computador, se recomienda que el primer paso a ejecutar sea el comando `make clean`, ubicado en el directorio raíz.

1.2.2 2. Compilación del Programa

Para compilar el proyecto completo, ejecute el comando `make` en el directorio raíz del proyecto. Esto utilizará el archivo `Makefile`.

1.2.3 3. Limpieza del Historial

El proyecto cuenta con un historial para poder guardar los usuarios anteriormente creados, este historial está límitado a `50 usuarios`, por ende, si quiere seguir utilizando el programa desde cero, debe ejecutar el comando `make clear-log`, esto hará que se limpie el archivo txt que contiene el historial.

1.2.4 4. Ejecución

Para poder ejecutar el funcionamiento del proyecto debe utilizar el comando `make run`, esto utiliza el archivo `Makefile`, dentro de este se especifica la cantidad de usuarios que genera, si quiere cambiar la cantidad, debe ingresar al archivo `Makefile` y cambiar el valor allí.

1.2.5 5. Documentación

Ejecute el comando `make dxygn` en el directorio raíz del proyecto para generar la documentación automática en formato HTML y LaTeX.

Una vez generados los archivos con Doxygen con el comando `make dxygn`, puede convertir la documentación en un archivo PDF utilizando LaTeX. Para hacerlo, ejecute `make ltx`.

1.2.6 6. Ubicación de los archivos PDF y HTML

El archivo PDF generado estará disponible en el directorio docs/latex/ o en la ubicación configurada en el `Makefile`. El archivo se llama `refman.pdf`.

Los archivos HTML generados se encontrarán en el directorio docs/html/. Para visualizar la documentación, abra el archivo index.html con su navegador. El archivo se llama `index.html`.

1.2.7 7. Ubicación del Gráfico y Archivos TXT

El gráfico generado por el programa al final de la ejecución se encuentra en el directorio `output` (visto desde el directorio raíz).

Los archivos .txt que contienen las listas de nombres, hobbies, personalidades e historial, se encuentran en el directorio `input` (visto desde el directorio raíz).

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Graph Struct Reference

Estructura de un grafo.

`#include <graph.h>`

Public Attributes

- int numUsers
- Node ** adjacencyList
- char ** user_names

4.1.1 Detailed Description

Estructura de un grafo.

```
typedef struct Graph
{
     int numUsers;
     Node **adjacencyList;
     char **user_names;
} Graph;
```

Definition at line 45 of file graph.h.

4.1.2 Member Data Documentation

4.1.2.1 adjacencyList

`Node** Graph::adjacencyList`

Definition at line 48 of file graph.h.

4.1.2.2 numUsers

```
int Graph::numUsers
```

Definition at line 47 of file graph.h.

4.1.2.3 user_names

```
char** Graph::user_names
```

Definition at line 49 of file graph.h.

The documentation for this struct was generated from the following file:

- incs/graph.h

4.2 Match Struct Reference

Estructura que representa una coincidencia entre dos usuarios. Esta estructura se utiliza para almacenar la similitud de hobbies y la diferencia de edad entre dos usuarios que se han emparejado en el sistema.

```
#include <users.h>
```

Public Attributes

- int user_index
- double similarity
- int age_diff

4.2.1 Detailed Description

Estructura que representa una coincidencia entre dos usuarios. Esta estructura se utiliza para almacenar la similitud de hobbies y la diferencia de edad entre dos usuarios que se han emparejado en el sistema.

```
typedef struct
{
    int user_index;
    double similarity;
    int age_diff;
} Match;
```

Definition at line 80 of file users.h.

4.2.2 Member Data Documentation

4.2.2.1 age_diff

```
int Match::age_diff
```

Definition at line 84 of file users.h.

4.2.2.2 similarity

```
double Match::similarity
```

Definition at line 83 of file users.h.

4.2.2.3 user_index

```
int Match::user_index
```

Definition at line 82 of file users.h.

The documentation for this struct was generated from the following file:

- incs/users.h

4.3 Node Struct Reference

Estructura de un nodo.

```
#include <graph.h>
```

Public Attributes

- int id
- int weight
- struct Node * next

4.3.1 Detailed Description

Estructura de un nodo.

```
typedef struct Node
{
     int id;
     int weight;
     struct Node *next;
} Node;
```

Definition at line 26 of file graph.h.

4.3.2 Member Data Documentation

4.3.2.1 id

```
int Node::id
```

Definition at line 28 of file graph.h.

4.3.2.2 next

```
struct Node* Node::next
```

Definition at line 30 of file graph.h.

4.3.2.3 weight

```
int Node::weight
```

Definition at line 29 of file graph.h.

The documentation for this struct was generated from the following file:

- incs/graph.h

4.4 Post Struct Reference

Estructura que representa una publicación de un usuario. Esta estructura contiene la información relacionada con una publicación, incluyendo un identificador único, el ID del usuario que la crea, el contenido de la publicación, una marca de tiempo, y un puntero al siguiente post en una lista enlazada.

```
#include <posts.h>
```

Public Attributes

- int post_Id
- int user_Id
- char username [MAX_NAME_LENGTH]
- char content [MAX_POST_LENGTH]
- time_t timestamp
- struct Post * next

4.4.1 Detailed Description

Estructura que representa una publicación de un usuario. Esta estructura contiene la información relacionada con una publicación, incluyendo un identificador único, el ID del usuario que la crea, el contenido de la publicación, una marca de tiempo, y un puntero al siguiente post en una lista enlazada.

```
typedef struct Post
{
    int post_Id;
    int user_Id;
    char username[MAX_NAME_LENGTH];
    char content[MAX_POST_LENGTH];
    time_t timestamp;
    struct Post *next;
} Post;
```

Definition at line 46 of file posts.h.

4.4.2 Member Data Documentation

4.4.2.1 content

```
char Post::content[MAX_POST_LENGTH]
```

Definition at line 51 of file posts.h.

4.4.2.2 next

```
struct Post* Post::next
```

Definition at line 53 of file posts.h.

4.4.2.3 post_Id

```
int Post::post_Id
```

Definition at line 48 of file posts.h.

4.4.2.4 timestamp

```
time_t Post::timestamp
```

Definition at line 52 of file posts.h.

4.4.2.5 user_Id

```
int Post::user_Id
```

Definition at line 49 of file posts.h.

4.4.2.6 username

```
char Post::username[MAX_NAME_LENGTH]
```

Definition at line 50 of file posts.h.

The documentation for this struct was generated from the following file:

- incs/posts.h

4.5 Post_List Struct Reference

Estructura que representa una lista de publicaciones. Esta estructura contiene un puntero a la cabeza de la lista de publicaciones y el contador de publicaciones.

```
#include <posts.h>
```

Public Attributes

- Post * head
- int postCount

4.5.1 Detailed Description

Estructura que representa una lista de publicaciones. Esta estructura contiene un puntero a la cabeza de la lista de publicaciones y el contador de publicaciones.

```
typedef struct Post_List
{
     Post *head;
     int postCount;
} Post_List;
```

Definition at line 69 of file posts.h.

4.5.2 Member Data Documentation

4.5.2.1 head

```
Post* Post_List::head
```

Definition at line 71 of file posts.h.

4.5.2.2 postCount

```
int Post_List::postCount
```

Definition at line 72 of file posts.h.

The documentation for this struct was generated from the following file:

- incs/posts.h

4.6 User Struct Reference

Estructura que representa a un usuario. Esta estructura contiene la información básica de un usuario, como su ID, nombre, edad, género, hobbies y personalidad.

```
#include <users.h>
```

Public Attributes

- int id
- char username [MAX_NAME_LENGTH]
- int age
- char gender [MAX_GENDER]
- char hobbies [MAX_HOBBIES][MAX_HOBBIE_LENGTH]
- char personality [MAX_PERS_LENGTH]

4.6.1 Detailed Description

Estructura que representa a un usuario. Esta estructura contiene la información básica de un usuario, como su ID, nombre, edad, género, hobbies y personalidad.

```
typedef struct User
{
     int id;
     char username[MAX_NAME_LENGTH];
     int age;
     char gender[MAX_GENDER];
     char hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH];
     char personality[MAX_PERS_LENGTH];
} User;
```

Definition at line 56 of file users.h.

4.6.2 Member Data Documentation

4.6.2.1 age

```
int User::age
```

Definition at line 60 of file users.h.

4.6.2.2 gender

```
char User::gender[MAX_GENDER]
```

Definition at line 61 of file users.h.

4.6.2.3 hobbies

```
char User::hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH]
```

Definition at line 62 of file users.h.

4.6.2.4 id

```
int User::id
```

Definition at line 58 of file users.h.

4.6.2.5 personality

```
char User::personality[MAX_PERS_LENGTH]
```

Definition at line 63 of file users.h.

4.6.2.6 username

```
char User::username[MAX_NAME_LENGTH]
```

Definition at line 59 of file users.h.

The documentation for this struct was generated from the following file:

- incs/users.h

Chapter 5

File Documentation

5.1 incs/graph.h File Reference

Prototipos de funciones para la conexion entre usuarios (amistad)

```
#include "users.h"
```

Classes

- struct Node

  Estructura de un nodo.
- struct Graph

  Estructura de un grafo.

Typedefs

- typedef struct Node **Node**
- typedef struct Graph **Graph**

Functions

- Graph ∗ initialize_graph (int, User ∗)

  Inicializa un grafo con un número específico de usuarios y sus nombres. Reserva memoria y configura una estructura de grafo, inicializando la lista de adyacencia y copiando los nombres de los usuarios.
- void add_connection (Graph ∗, int, int)

  Agrega una conexión bidireccional entre dos usuarios en el grafo. Inserta nodos en la lista de adyacencia para conectar a dos usuarios,representando una relación de amistad.
- void display_graph (Graph ∗, int)

  Muestra las conexiones del grafo utilizando el algoritmo de Dijkstra. Calcula las rutas más cortas desde un usuario específico al resto utilizando Dijkstra y muestra la distancia y el camino hacia el nodo más lejano alcanzable.
- void print_path (int, int ∗, Graph ∗)

  Imprime el camino desde un nodo fuente a un nodo destino. Muestra la secuencia de nombres de usuarios que forman la ruta más corta calculada previamente.
- void free_graph (Graph ∗)

Libera toda la memoria asociada con el grafo. Elimina todas las estructuras dinámicas asociadas con la lista de adyacencia y los nombres de los usuarios.

- void create_connections (const User users[MAX_USERS], int, Graph *, double)

  Genera conexiones aleatorias entre usuarios en el grafo.
- int find_user_with_most_friends (Graph *)

  Encuentra el usuario con más amigos en el grafo.
- void print_friends_of_user (Graph *, int)

  Encuentra el usuario con menos amigos en el grafo.
- void generate_eps_graph (Graph *, const char *)

  Genera un archivo .dot con la representación del grafo.
- void transform_eps_png (const char *)

  Transforma un archivo .eps a .png.

5.1.1 Detailed Description

Prototipos de funciones para la conexion entre usuarios (amistad)

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Contiene los prototipos de las funciones dedicadas a la creación de conexiones entre usuarios

Definition in file graph.h.

5.1.2 Function Documentation

5.1.2.1 add_connection()

```
void add_connection (
            Graph * graph,
            int user1,
            int user2)
```

Agrega una conexión bidireccional entre dos usuarios en el grafo. Inserta nodos en la lista de adyacencia para conectar a dos usuarios,representando una relación de amistad.

Parameters

graph	Puntero al grafo donde se desea agregar la conexión.
user1	Identificador del primer usuario.
user2	Identificador del segundo usuario.

Agrega una conexión bidireccional entre dos usuarios en el grafo. Inserta nodos en la lista de adyacencia para conectar a dos usuarios,representando una relación de amistad.

Parameters

graph	Puntero al grafo donde se agregarán las conexiones.
user1	Identificador del primer usuario.
user2	Identificador del segundo usuario.

Agrega una conexión bidireccional entre dos usuarios en el grafo.

```
if (!graph) ->error
Node *newNode = (Node *)malloc(sizeof(Node));
newNode->id = user2;
newNode->weight = 1;
newNode->next = graph->adjacencyList[user1];
graph->adjacencyList[user1] = newNode;
newNode = (Node *)malloc(sizeof(Node));
newNode->id = user1;
newNode->weight = 1;
newNode->next = graph->adjacencyList[user2];
graph->adjacencyList[user2] = newNode;
```

Definition at line 55 of file connections_graph.c.

5.1.2.2 create_connections()

```
void create_connections (
            const User users[MAX_USERS],
            int num_users,
            Graph * graph,
            double threshold)
```

Genera conexiones aleatorias entre usuarios en el grafo.

Parameters

users	Arreglo de usuarios disponibles.
numUsers	Número total de usuarios.
graph	Puntero al grafo donde se generarán las conexiones.
density	Densidad de conexiones (valor entre 0 y 1).

Genera conexiones aleatorias entre usuarios en el grafo.

Parameters

users	Arreglo de usuarios a evaluar.
num_users	Número total de usuarios.
graph	Grafo donde se almacenarán las conexiones.
threshold	Valor mínimo de similitud de Jaccard para crear una conexión.

Crea conexiones entre usuarios que tienen un índice de similitud de Jaccard por encima de un umbral.

```
int connections_found = 0;
for (int i = 0; i < num_users; i++)
  for (int j = i + 1; j < num_users; j++)
       int count1 = 0, count2 = 0;
       double similarity = calculate_jaccard_similarity(users[i].hobbies, count1, users[j].hobbies, count2,
      users[i].age, users[j].age, users[i].personality, users[j].personality);
       if (similarity >= threshold)
          connections_found = 1;
          fprintf(stdout, CYAN "\nConectando a los usuarios %s y %s (Índice de Jaccard: %.2f)\n" RESET,
      users[i].username, users[j].username, similarity);
          add_connection(graph, i, j);
if (!connections_found) ->error
```

Definition at line 178 of file similarity.c.

5.1.2.3 display_graph()

```
void display_graph (
            Graph * graph,
            int source)
```

Muestra las conexiones del grafo utilizando el algoritmo de Dijkstra. Calcula las rutas más cortas desde un usuario específico al resto utilizando Dijkstra y muestra la distancia y el camino hacia el nodo más lejano alcanzable.

Parameters

graph	Puntero al grafo.
source	Identificador del usuario desde el cual calcular las rutas.

Muestra las conexiones del grafo utilizando el algoritmo de Dijkstra. Calcula las rutas más cortas desde un usuario específico al resto utilizando Dijkstra y muestra la distancia y el camino hacia el nodo más lejano alcanzable.

Parameters

graph	Puntero al grafo cuyas conexiones se mostrarán.
source	Identificador del nodo fuente desde el cual calcular las distancias.

Muestra las conexiones del grafo y el camino más largo desde un nodo fuente utilizando Dijkstra.

```
double distance[MAX_USERS];
int prev_node[MAX_USERS];
int visited_node[MAX_USERS] = {0};
for(int i=0;i<graph->numUsers;i++)
  Aqui se inicializan las distancias de los nodos desde el nodo fuente y cual es el nodo previo
distance[source]=0;
for(int i=0;i<graph->numUsers;i++)
   Algoritmo Dijkstra
int farthest_node=-1;
int max_distance=-1;
```

Definition at line 96 of file connections_graph.c.

5.1.2.4 find_user_with_most_friends()

```
int find_user_with_most_friends (
            Graph * graph)
```

Encuentra el usuario con más amigos en el grafo.

Parameters

graph	Puntero al grafo.

Returns

Identificador del usuario con más amigos.

Encuentra el usuario con más amigos en el grafo.

Parameters

graph	Puntero al grafo de usuarios donde se buscan los amigos.

Returns

El índice del usuario con más amigos. Si no se encuentra ningún usuario, devuelve -1.

Encuentra al usuario con más amigos en el grafo.

```
int maxFriends = -1;
int userIndex = -1;
int i;
if (!graph || graph->numUsers == 0) ->error
for (i = 0; i < graph->numUsers; i++)
     int friendCount = 0;
     Node *current = graph->adjacencyList[i];
     while (current)
         Contar las conexiones del usuario actual
     if (friendCount > maxFriends)
         Contar las conexiones del usuario actual
     if (friendCount > maxFriends)
         Actualizar si el usuario actual tiene más amigos
if (userIndex != -1)->muestra usuario con mas amigos
else -> no se encontraron usuarios con amigos
```

Definition at line 15 of file search.c.

5.1.2.5 free_graph()

```
void free_graph (
             Graph * graph)
```

Libera toda la memoria asociada con el grafo. Elimina todas las estructuras dinámicas asociadas con la lista de adyacencia y los nombres de los usuarios.

Parameters

graph	Puntero al grafo a liberar.

Libera toda la memoria asociada con el grafo. Elimina todas las estructuras dinámicas asociadas con la lista de adyacencia y los nombres de los usuarios.

Parameters

graph	Puntero al grafo a liberar.

Libera toda la memoria asignada al grafo

```
for (int i = 0; i < graph->numUsers; i++)
  libera nombres de usuarios
free(graph->user_names);
free(graph->adjacencyList);
free(graph);
```

Definition at line 205 of file connections_graph.c.

5.1.2.6 generate_eps_graph()

```
void generate_eps_graph (
             Graph * graph,
             const char * filename)
```

Genera un archivo .dot con la representación del grafo.

Parameters

graph	Puntero al grafo.
filename	Nombre del archivo .dot a generar.

Genera un archivo .dot con la representación del grafo.

Parameters

graph	Puntero al grafo que se desea visualizar.
filename	Nombre del archivo EPS a generar.

Note

Si el grafo o el nombre del archivo no son válidos, el programa finaliza con error.

Genera un archivo EPS que representa un grafo y lo convierte a PNG.

```
if (!graph || !filename) ->error
FILE *file = fopen(filename, "w");
if (!file) ->error
int radius = 200;
int centerX = 250;
int centerY = 250;
double angleStep = 2 * M_PI / graph->numUsers;
int positions[MAX_USERS][2];
for (int i = 0; i < graph->numUsers; i++)
  Almacena las posiciones de cada nodo.
for (int i = 0; i < graph->numUsers; i++)
  Dibujar nodos (usuarios) y nombres de cada uno.
transform_eps_png(filename);
```

Definition at line 15 of file graphic.c.

5.1.2.7 initialize_graph()

```
Graph * initialize_graph (
            int numUsers,
            User * users)
```

Inicializa un grafo con un número específico de usuarios y sus nombres. Reserva memoria y configura una estructura de grafo, inicializando la lista de adyacencia y copiando los nombres de los usuarios.

Parameters

numUsers	Número total de usuarios en el grafo.
users	Puntero al arreglo de usuarios, cada uno con su nombre.

Returns

Puntero al grafo inicializado.

Inicializa un grafo con un número específico de usuarios y sus nombres. Reserva memoria y configura una estructura de grafo, inicializando la lista de adyacencia y copiando los nombres de los usuarios.

Parameters

numUsers	Número total de usuarios en el grafo.
users	Puntero al arreglo de usuarios, cada uno con su nombre.

Returns

Puntero al grafo inicializado.

Inicializa un grafo con un número específico de usuarios y sus nombres.

```
if (numUsers <= 0) ->error
Graph *graph = (Graph *)malloc(sizeof(Graph));
graph->numUsers = numUsers;
graph->adjacencyList = (Node **)calloc(numUsers + 1, sizeof(Node *));
graph->user_names = malloc(numUsers * sizeof(char *));
for (int i = 0; i < numUsers; i++)
  Agregar nombres de usuarios.
return graph;
```

Definition at line 15 of file connections_graph.c.

5.1.2.8 print_friends_of_user()

```
void print_friends_of_user (
            Graph * graph,
            int userIndex)
```

Encuentra el usuario con menos amigos en el grafo.

Parameters

graph	Puntero al grafo.

Returns

Identificador del usuario con menos amigos.

Encuentra el usuario con menos amigos en el grafo.

Parameters

graph	Puntero al grafo que contiene la información de los usuarios y sus amigos.
userIndex	Índice del usuario en el grafo del cual se desean imprimir los amigos.

Imprime los amigos de un usuario dado en el grafo.

```
if (!graph || userIndex < 0 || userIndex >= graph->numUsers) ->error
Node *current = graph->adjacencyList[userIndex];
fprintf(stdout, CYAN "Amigos de %s:\n" RESET, graph->user_names[userIndex]);
while (current)
  imprime los amigos del usuario con mas amigos
```

Definition at line 79 of file search.c.

5.1.2.9 print_path()

```
void print_path (
            int target,
            int * prev_node,
            Graph * graph)
```

Imprime el camino desde un nodo fuente a un nodo destino. Muestra la secuencia de nombres de usuarios que forman la ruta más corta calculada previamente.

Parameters

target	Identificador del nodo destino.
previous	Arreglo con los nodos previos en el camino calculado por Dijkstra.
graph	Puntero al grafo que contiene los nombres de los usuarios.

Imprime el camino desde un nodo fuente a un nodo destino. Muestra la secuencia de nombres de usuarios que forman la ruta más corta calculada previamente.

Parameters

target	Nodo objetivo.
prev_node	Arreglo con los predecesores de cada nodo.
graph	Puntero al grafo.

Muestra el camino desde un nodo fuente hasta un objetivo.

```
if (prev_node[target] == -1)
  fprintf(stdout, "%s", graph->user_names[target]);
  return;
print_path(prev_node[target], prev_node, graph);
fprintf(stdout, " -> %s", graph->user_names[target]);
```

Definition at line 181 of file connections_graph.c.

5.1.2.10 transform_eps_png()

```
void transform_eps_png (
            const char * filename)
```

Transforma un archivo .eps a .png.

Parameters

filename	Nombre del archivo .eps a transformar.

Transforma un archivo .eps a .png.

Parameters

filename	Nombre del archivo EPS a convertir.

Note

Si el nombre del archivo no es válido o la conversión falla, el programa finaliza con error.

Convierte un archivo EPS a PNG y elimina el archivo EPS.

```
if (!filename) ->error
char base_filename[256];
strncpy(base_filename, filename, sizeof(base_filename) - 1);
base_filename[sizeof(base_filename) - 1] = '\0';
char *ext = strrchr(base_filename, '.');
if (ext && strcmp(ext, ".eps") == 0)
  Eliminar la extensión .eps
 char command[512];
snprintf(command, sizeof(command), "gs -dSAFER -dBATCH -dNOPAUSE -dEPSCrop -sDEVICE=png16m -r300
      -sOutputFile=%s.png %s > /dev/null 2>&1", base_filename, filename);
int result = system(command);
if (result != 0) ->error
if (remove(filename) != 0)
  Eliminar el archivo EPS.
```

Definition at line 110 of file graphic.c.

5.2 graph.h

Go to the documentation of this file.

```
00001 #ifndef GRAPH_H
00002 #define GRAPH_H
00003
00012 #include "users.h"
00013
00026 typedef struct Node
00027 {
00028     int id;
00029     int weight;
00030     struct Node *next;
00031 } Node;
00032
00045 typedef struct Graph
00046 {
00047     int numUsers;
00048     Node **adjacencyList;
00049     char **user_names;
00050 } Graph;
00051
00059 Graph *initialize_graph(int, User *);
00060
00068 void add_connection(Graph *, int, int);
00069
00076 void display_graph(Graph *, int);
00077
00085 void print_path(int, int *, Graph *);
00086
00092 void free_graph(Graph *);
00093
00101 void create_connections(const User users[MAX_USERS], int, Graph *, double);
00102
00108 int find_user_with_most_friends(Graph *);
00109
00115 void print_friends_of_user(Graph *, int);
00116
00122 void generate_eps_graph(Graph *, const char *);
00123
00128 void transform_eps_png(const char *);
00129
00130 #endif
```

5.3 incs/log.h File Reference

Definición de funciones para manejar el historial de usuarios y registros.

```
#include "users.h"
```

Functions

- void user_count_from_log (int *)

  Función que cuenta el número de usuarios registrados en el archivo de logs.
- int log_check ()

  Función que verifica el estado del archivo de logs.
- void log_input (User *)

  Función que registra la entrada de un usuario en el archivo de log.
- void log_clean ()

  Función que limpia el archivo de logs, eliminando todos los registros de usuarios.
- void log_output (const User *)

  Función que imprime la información de un usuario desde el log.

5.3.1 Detailed Description

Definición de funciones para manejar el historial de usuarios y registros.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Contiene los prototipos de las funciones dedicadas a la manipulación del archivo de historial.

Definition in file log.h.

5.3.2 Function Documentation

5.3.2.1 log_check()

```
int log_check ()
```

Función que verifica el estado del archivo de logs.

Returns

Retorna 1 si el archivo de log es accesible, 0 si no lo es.

Función que verifica el estado del archivo de logs.

Returns

1 si hay historial, 0 si no lo hay.

Verifica si el archivo de log contiene registros.

```
FILE *file = fopen("./input/users_log.txt", "r");
if (file == NULL) -> Error
int char_file = fgetc(file);
fclose(file);
return (char_file != EOF);
```

Definition at line 51 of file user_log.c.

5.3.2.2 log_clean()

```
void log_clean ()
```

Función que limpia el archivo de logs, eliminando todos los registros de usuarios.

Función que limpia el archivo de logs, eliminando todos los registros de usuarios. Limpia el archivo de log de usuarios.

```
FILE *file = fopen("./input/users_log.txt", "w");
if (file == NULL) -> Error
fclose(file);
```

Definition at line 144 of file user_log.c.

5.3.2.3 log_input()

```
void log_input (
            User * )
```

Función que registra la entrada de un usuario en el archivo de log.

Parameters

user	Puntero al usuario que se está registrando.

5.3.2.4 log_output()

```
void log_output (
            const User * user)
```

Función que imprime la información de un usuario desde el log.

Parameters

user	Puntero al usuario cuya información se imprimirá.

Función que imprime la información de un usuario desde el log.

Parameters

user	Puntero al usuario que se desea agregar al archivo de log.

Función que agrega un nuevo usuario al archivo de log.

```
FILE *file = fopen("./input/users_log.txt", "a");
if (!file) -> Error
fprintf(file, "ID: %d\n", user->id);
fprintf(file, "Nombre: %s\n", user->username);
fprintf(file, "Género: %s\n", user->
fprintf(file, "Edad: %d\n", user->age);
fprintf(file, "Personalidad: %s\n", user->personality);
for (int i = 0; i < MAX_HOBBIES && user->hobbies[i][0] != '\0'; i++)
    fprintf(file, " - %s\n", user->hobbies[i]);
fprintf(file, "---\n");
fclose(file);
```

Definition at line 170 of file user_log.c.

5.3.2.5 user_count_from_log()

```
void user_count_from_log (
            int * user_count)
```

Función que cuenta el número de usuarios registrados en el archivo de logs.

Parameters

count	Puntero a la variable donde se almacenará el número de usuarios.

Función que cuenta el número de usuarios registrados en el archivo de logs.

Parameters

user_count	Puntero a la variable que almacenará el número de usuarios encontrados.

Cuenta el número de usuarios en el archivo de log.

```
FILE *file = fopen("./input/users_log.txt", "r");
const char *key_word = "ID:";
char buffer[100];
if (file == NULL) -> Error
while (fscanf(file, "%99s", buffer) == 1)
    if (strcasecmp(buffer, key_word) == 0)
       (*user_count)++;
fclose(file);
```

Definition at line 14 of file user_log.c.

5.4 log.h

Go to the documentation of this file.

```
00001 #ifndef LOG_H
00002 #define LOG_H
00003
00012 #include "users.h"
00013
00018 void user_count_from_log(int *);
00019
00024 int log_check();
00025
00030 void log_input(User *);
00031
00035 void log_clean();
00036
00041 void log_output(const User *);
00042
00043 #endif
```

5.5 incs/main.h File Reference

Definición de las librerías necesarias y macros de color para el proyecto.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include "users.h"
#include "posts.h"
#include "graph.h"
#include "log.h"
#include "similarity.h"
```

Macros

- #define RESET "\033[0m"
- #define GREEN "\033[32m"
- #define YELLOW "\033[33m"
- #define CYAN "\033[36m"
- #define RED "\033[31m"

5.5.1 Detailed Description

Definición de las librerías necesarias y macros de color para el proyecto.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Contiene las librerías necesarias para el proyecto y macros de color para la consola.

Definition in file main.h.

5.5.2 Macro Definition Documentation

5.5.2.1 CYAN

```
#define CYAN "\033[36m"
```

Definition at line 60 of file main.h.

5.5.2.2 GREEN

```
#define GREEN "\033[32m"
```

Definition at line 58 of file main.h.

5.5.2.3 RED

```
#define RED "\033[31m"
```

Definition at line 61 of file main.h.

5.5.2.4 RESET

```
#define RESET "\033[0m"
```

Librerías utilizadas en el proyecto.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
```

Librerías propias del proyecto.

```
#include "users.h"
#include "posts.h"
#include "graph.h"
#include "log.h"
#include "similarity.h"
```

Macros utilizadas en el proyecto.

```
#define RESET "\033[0m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define CYAN "\033[36m"
#define RED "\033[31m"
```

Definition at line 57 of file main.h.

5.5.2.5 YELLOW

```
#define YELLOW "\033[33m"
```

Definition at line 59 of file main.h.

5.6 main.h

Go to the documentation of this file.

```
00001 #ifndef MAIN_H
00002 #define MAIN_H
00003
00024 #include <math.h>
00025 #include <stdlib.h>
00026 #include <stdio.h>
00027 #include <unistd.h>
00028 #include <string.h>
00029 #include <time.h>
00030
00041 #include "users.h"
00042 #include "posts.h"
00043 #include "graph.h"
00044 #include "log.h"
00045 #include "similarity.h"
00046
00057 #define RESET "\033[0m"
00058 #define GREEN "\033[32m"
00059 #define YELLOW "\033[33m"
00060 #define CYAN "\033[36m"
00061 #define RED "\033[31m"
00062
00063 #endif
```

5.7 incs/posts.h File Reference

Definición de funciones y estructuras para manejar las publicaciones de usuarios.

```
#include "users.h"
```

Classes

- struct Post

  Estructura que representa una publicación de un usuario. Esta estructura contiene la información relacionada con una publicación, incluyendo un identificador único, el ID del usuario que la crea, el contenido de la publicación, una marca de tiempo, y un puntero al siguiente post en una lista enlazada.
- struct Post_List

  Estructura que representa una lista de publicaciones. Esta estructura contiene un puntero a la cabeza de la lista de publicaciones y el contador de publicaciones.

Macros

- #define MAX_POST 10
- #define MAX_POST_LENGTH 256
- #define MAX_FILE_LINES 100
- #define MAX_POSTS 3

Typedefs

- typedef struct Post **Post**
- typedef struct Post_List **Post_List**

Functions

- void init_post_list (Post_List *)

  Inicializa una lista de publicaciones. Esta función inicializa la lista de publicaciones configurando el puntero `head` *a NULL y el contador postCount` a 0.*
- Post * create_post (int, const char *, const char *)

  Crea una nueva publicación. Esta función crea una nueva publicación, asignando un ID único, y copiando el nombre de usuario y contenido proporcionado. La marca de tiempo es generada al momento de la creación.
- void publish_post (Post_List *, const User *, const char *)

  Publica un nuevo post en la lista. Esta función agrega una nueva publicación al final de la lista de publicaciones.
- void display_all_posts (const Post_List *)

  Muestra todas las publicaciones en la lista. Esta función imprime el contenido de todas las publicaciones almacenadas en la lista.
- void free_all_posts (Post_List *)

  Libera la memoria ocupada por todas las publicaciones de la lista. Esta función recorre la lista de publicaciones y libera la memoria de cada publicación.
- void load_post_templates (char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH], int *)

  Carga plantillas de publicaciones desde un archivo. Esta función carga las plantillas de publicaciones desde un archivo de texto, donde cada línea corresponde a una plantilla.
- void generate_random_posts (User users[MAX_USERS], int, Post_List *)

  Genera publicaciones aleatorias para un conjunto de usuarios. Esta función genera un número de publicaciones aleatorias para los usuarios en el arreglo `users`*.*
- time_t generate_random_timestamp ()

  Genera una marca de tiempo aleatoria. Esta función genera una marca de tiempo aleatoria para simular el momento de publicación de una publicación.

5.7.1 Detailed Description

Definición de funciones y estructuras para manejar las publicaciones de usuarios.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Contiene los prototipos de las funciones dedicadas a la creación, publicación y visualización de publicaciones.

Definition in file posts.h.

5.7.2 Macro Definition Documentation

5.7.2.1 MAX_FILE_LINES

```
#define MAX_FILE_LINES 100
```

Definition at line 25 of file posts.h.

5.7.2.2 MAX_POST

```
#define MAX_POST 10
```

Macros utilizadas en el proyecto.
```
#define MAX_POST 10
#define MAX_POST_LENGTH 256
#define MAX_FILE_LINES 100
#define MAX_POSTS 3
```

Definition at line 23 of file posts.h.

5.7.2.3 MAX_POST_LENGTH

```
#define MAX_POST_LENGTH 256
```

Definition at line 24 of file posts.h.

5.7.2.4 MAX_POSTS

```
#define MAX_POSTS 3
```

Definition at line 26 of file posts.h.

5.7.3 Function Documentation

5.7.3.1 create_post()

```
Post * create_post (
            int user_Id,
            const char * username,
            const char * content)
```

Crea una nueva publicación. Esta función crea una nueva publicación, asignando un ID único, y copiando el nombre de usuario y contenido proporcionado. La marca de tiempo es generada al momento de la creación.

Parameters

userId	ID del usuario que publica.
username	Nombre del usuario.
content	Contenido de la publicación.

Returns

Un puntero a la nueva publicación creada.

Crea una nueva publicación. Esta función crea una nueva publicación, asignando un ID único, y copiando el nombre de usuario y contenido proporcionado. La marca de tiempo es generada al momento de la creación.

Parameters

user_Id	ID del usuario que realiza la publicación.
username	Nombre de usuario del autor de la publicación.
content	Contenido textual de la publicación.

Returns

Un puntero a la nueva publicación creada.

Crea una nueva publicación,esta función genera una nueva publicación con un ID único.

```
if (strlen(content) >= MAX_POST_LENGTH) ->error
Post *newPost = malloc(sizeof(Post));
if (!newPost) ->error
static int post_Id_Counter = 1;
newPost->post_Id = post_Id_Counter++;
newPost->user_Id = user_Id;
strncpy(newPost->username, username, MAX_NAME_LENGTH - 1);
newPost->username[MAX_NAME_LENGTH - 1] = '\0';
strncpy(newPost->content, content, MAX_POST_LENGTH - 1);
newPost->content[MAX_POST_LENGTH - 1] = '\0';
newPost->timestamp = generate_random_timestamp();
newPost->next = NULL;
```

Definition at line 33 of file post.c.

5.7.3.2 display_all_posts()

```
void display_all_posts (
            const Post_List * post_list)
```

Muestra todas las publicaciones en la lista. Esta función imprime el contenido de todas las publicaciones almacenadas en la lista.

Parameters

postList	Puntero a la lista de publicaciones.

Muestra todas las publicaciones en la lista. Esta función imprime el contenido de todas las publicaciones almacenadas en la lista.

Parameters

post_list	Puntero a la lista de publicaciones que se mostrarán.

Muestra todas las publicaciones en la red social. Imprime en consola la lista completa de publicaciones, mostrando la ID, el usuario, el contenido y la fecha.

```
fprintf(stdout, RED "\nPublicaciones:\n\n" RESET);
Post *current = post_list->head;
while (current)
   Imprime en consola la lista completa de publicaciones, mostrando la ID, el usuario, el contenido y la
      fecha.
```

Definition at line 110 of file post.c.

5.7.3.3 free_all_posts()

```
void free_all_posts (
            Post_List * post_list)
```

Libera la memoria ocupada por todas las publicaciones de la lista. Esta función recorre la lista de publicaciones y libera la memoria de cada publicación.

Parameters

postList	Puntero a la lista de publicaciones.

Libera la memoria ocupada por todas las publicaciones de la lista. Esta función recorre la lista de publicaciones y libera la memoria de cada publicación.

Parameters

post_list	Puntero a la lista de publicaciones que se deben liberar.

Libera la memoria ocupada por todas las publicaciones. Recorre la lista de publicaciones y libera la memoria de cada una de ellas.

```
Post *current = post_list->head;
while (current)
  Libera la memoria ocupada
post_list->head = NULL;
post_list->postCount = 0;
```

Definition at line 143 of file post.c.

5.7.3.4 generate_random_posts()

```
void generate_random_posts (
            User users[MAX_USERS],
            int num_users,
            Post_List * post_list)
```

Genera publicaciones aleatorias para un conjunto de usuarios. Esta función genera un número de publicaciones aleatorias para los usuarios en el arreglo `users`.

Parameters

users	Arreglo de usuarios para generar las publicaciones.
numPosts	Número de publicaciones a generar.
postList	Puntero a la lista de publicaciones.

Genera publicaciones aleatorias para un conjunto de usuarios. Esta función genera un número de publicaciones aleatorias para los usuarios en el arreglo `users`.

Parameters

users	Arreglo de usuarios disponibles para generar publicaciones.
num_users	Número de usuarios disponibles.
post_list	Puntero a la lista de publicaciones donde se agregarán las nuevas publicaciones generadas.

Genera publicaciones aleatorias usando usuarios y plantillas.

```
char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH];
int post_template_count = 0;
load_post_templates(post_templates, &post_template_count);
if (post_template_count == 0) ->error
int used_users[MAX_USERS] = {0};
int unique_posts_created = 0;
while (unique_posts_created < MAX_POSTS && unique_posts_created < num_users)
  Generar publicaciones aleatorias hasta alcanzar el límite o usar todos los usuarios
```

Definition at line 220 of file post.c.

5.7.3.5 generate_random_timestamp()

```
time_t generate_random_timestamp ()
```

Genera una marca de tiempo aleatoria. Esta función genera una marca de tiempo aleatoria para simular el momento de publicación de una publicación.

Returns

La marca de tiempo aleatoria generada.

Genera una marca de tiempo aleatoria. Esta función genera una marca de tiempo aleatoria para simular el momento de publicación de una publicación.

Returns

El timestamp generado aleatoriamente.

Genera un timestamp aleatorio.

```
time_t current_time = time(NULL);
int random_hours = rand() % (24 * 7);
int random_minutes = rand() % 60;
int random_seconds = rand() % 60;
time_t random_time = current_time - (random_hours * 3600 + random_minutes * 60 + random_seconds);
return random_time;
```

Definition at line 272 of file post.c.

5.7.3.6 init_post_list()

```
void init_post_list (
            Post_List * post_list)
```

Inicializa una lista de publicaciones. Esta función inicializa la lista de publicaciones configurando el puntero `head` a NULL y el contador postCount` a 0.

Parameters

list	Puntero a la lista de publicaciones a inicializar.

Inicializa una lista de publicaciones. Esta función inicializa la lista de publicaciones configurando el puntero `head` a NULL y el contador postCount` a 0.

Parameters

post_list	Puntero a la lista de publicaciones a inicializar.

Inicializa la lista de publicaciones, configura el encabezado de la lista de publicaciones a NULL y establece el contador de publicaciones en 0.

```
post_list->head = NULL;
post_list->postCount = 0;
```

Definition at line 13 of file post.c.

5.7.3.7 load_post_templates()

```
void load_post_templates (
            char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH],
            int * post_count)
```

Carga plantillas de publicaciones desde un archivo. Esta función carga las plantillas de publicaciones desde un archivo de texto, donde cada línea corresponde a una plantilla.

Parameters

post_templates	Arreglo donde se almacenarán las plantillas de publicaciones.
count	Número de plantillas cargadas.

Carga plantillas de publicaciones desde un archivo. Esta función carga las plantillas de publicaciones desde un archivo de texto, donde cada línea corresponde a una plantilla.

Parameters

post_templates	Arreglo donde se almacenarán las plantillas de publicaciones.
post_count	Puntero a la variable que contará las plantillas cargadas.

Carga plantillas de publicaciones desde un archivo.

```
FILE *file = fopen("./input/post_templates.txt", "r");
if (!file) ->error
*post_count = 0;
char line[MAX_POST_LENGTH];
while (*post_count < MAX_FILE_LINES && fgets(line, sizeof(line), file))
  Carga plantillas de publicaciones desde un archivo.
```

Definition at line 174 of file post.c.

5.7.3.8 publish_post()

```
void publish_post (
            Post_List * post_list,
            const User * user,
            const char * content)
```

Publica un nuevo post en la lista. Esta función agrega una nueva publicación al final de la lista de publicaciones.

Parameters

postList	Puntero a la lista de publicaciones.
user	Puntero a la estructura del usuario que publica.
content	Contenido de la publicación.

Publica un nuevo post en la lista. Esta función agrega una nueva publicación al final de la lista de publicaciones.

Parameters

post_list	Puntero a la lista de publicaciones global.
user	Puntero a la estructura del usuario que realiza la publicación.
content	Contenido textual de la publicación.

Publica una nueva entrada en el muro global-Crea una publicación usando la información de un usuario y agrega la publicación al inicio de la lista global de publicaciones.

```
Post *newPost = create_post(user->id, user->username, content);
if (newPost)
  Agregar al inicio de la lista global
```

Definition at line 86 of file post.c.

5.8 posts.h

Go to the documentation of this file.

```
00001 #ifndef POSTS_H
00002 #define POSTS_H
00003
00012 #include "users.h"
00013
00023 #define MAX_POST 10
00024 #define MAX_POST_LENGTH 256
00025 #define MAX_FILE_LINES 100
00026 #define MAX_POSTS 3
00027
00046 typedef struct Post
00047 {
00048   int post_Id;
00049   int user_Id;
00050   char username[MAX_NAME_LENGTH];
00051   char content[MAX_POST_LENGTH];
00052   time_t timestamp;
00053   struct Post *next;
00054 } Post;
00055
00068 /* Estructura de Lista de Publicaciones */
00069 typedef struct Post_List
00070 {
00071   Post *head;
00072   int postCount;
00073 } Post_List;
00074
00081 void init_post_list(Post_List *);
00082
00092 Post *create_post(int, const char *, const char *);
00093
00101 void publish_post(Post_List *, const User *, const char *);
00102
00108 void display_all_posts(const Post_List *);
00109
00115 void free_all_posts(Post_List *);
00116
00123 void load_post_templates(char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH], int *);
00124
00132 void generate_random_posts(User users[MAX_USERS], int, Post_List *);
00133
00139 time_t generate_random_timestamp();
00140
00141 #endif
```

5.9 incs/similarity.h File Reference

Definición de funciones para calcular la similitud entre usuarios. Este archivo contiene funciones que permiten calcular la similitud entre usuarios basándose en sus hobbies, edad y personalidad. Además, incluye algoritmos para recomendar usuarios y ordenar las coincidencias de manera eficiente.

```
#include "users.h"
```

Functions

- double calculate_jaccard_similarity (const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int, const char hobbies2[MAX_HOBBIE_LENGTH][MAX_HOBBIE_LENGTH], int, int, int, const char *, const char *)

  Calcula la similitud de Jaccard entre dos conjuntos de hobbies. Esta función calcula la similitud de Jaccard entre los hobbies de dos usuarios, tomando en cuenta el número de hobbies comunes y el número total de hobbies.

- void find_common_hobbies (const char[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int, const char[MAX_↩ HOBBIES][MAX_HOBBIE_LENGTH], int)

  Encuentra los hobbies comunes entre dos usuarios. Esta función compara los hobbies de dos usuarios y encuentra aquellos que coinciden.

- void recommend_users (const User users[MAX_USERS], int)

  Recomienda usuarios basándose en la similitud. Compara todos los usuarios entre sí y recomienda aquellos con mayor compatibilidad basada en la similitud de hobbies, edad y personalidad.
- double calculate_age_weight (int, int)

  Calcula un factor de ponderación basado en la diferencia de edad entre dos usuarios. La diferencia de edad se convierte en un valor numérico para ajustar la similitud entre usuarios.
- const char * get_age_compatibility_level (int)

  Obtiene el nivel de compatibilidad de edad entre dos usuarios. Esta función clasifica la compatibilidad de edad entre dos usuarios en categorías.
- double calculate_personality_multiplier (int, int)

  Calcula un multiplicador de personalidad entre dos usuarios. Compara las personalidades de dos usuarios y devuelve un multiplicador que indica cuán compatibles son sus personalidades.
- void explain_personality_compatibility (const User *, const User *)

  Explica la compatibilidad de personalidad entre dos usuarios. Imprime una descripción detallada de cómo las personalidades de dos usuarios son compatibles.
- int get_personality_group (const char *)

  Obtiene el grupo de personalidad al que pertenece un usuario. La personalidad de un usuario se clasifica en un grupo de personalidades predefinido.
- void quicksort (Match matches[ ], int, int)

  Ordena un arreglo de coincidencias de usuarios utilizando el algoritmo QuickSort. Este algoritmo organiza las coincidencias de acuerdo con su similitud, de mayor a menor.
- int partition (Match matches[ ], int, int)

  Particiona un arreglo de coincidencias para el algoritmo QuickSort. Se utiliza para reorganizar el arreglo y preparar el pivote para la ordenación.

5.9.1 Detailed Description

Definición de funciones para calcular la similitud entre usuarios. Este archivo contiene funciones que permiten calcular la similitud entre usuarios basándose en sus hobbies, edad y personalidad. Además, incluye algoritmos para recomendar usuarios y ordenar las coincidencias de manera eficiente.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Definition in file similarity.h.

5.9.2 Function Documentation

5.9.2.1 calculate_age_weight()

```
double calculate_age_weight (
            int age1,
            int age2)
```

Calcula un factor de ponderación basado en la diferencia de edad entre dos usuarios. La diferencia de edad se convierte en un valor numérico para ajustar la similitud entre usuarios.

Parameters

age1	Edad del primer usuario.
age2	Edad del segundo usuario.

Returns

Un valor de ponderación basado en la diferencia de edad.

Calcula un factor de ponderación basado en la diferencia de edad entre dos usuarios. La diferencia de edad se convierte en un valor numérico para ajustar la similitud entre usuarios.

Parameters

age1	La edad del primer usuario.
age2	La edad del segundo usuario.

Returns

Un valor flotante que representa el peso de la compatibilidad basado en la diferencia de edad.

Calcula el peso de la compatibilidad basado en la diferencia de edad, dependiendo de la diferencia de edad, se aplica un factor de penalización.

```
int age_diff = abs(age1 - age2);
if (age_diff <= 5)
  return 1.0;
else if (age_diff <= 10)
  return 0.8;
else if (age_diff <= 15)
  return 0.6;
else if (age_diff <= 20)
  return 0.4;
else
  return 0.2;
```

Definition at line 149 of file parameters.c.

5.9.2.2 calculate_jaccard_similarity()

```
double calculate_jaccard_similarity (
            const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            int count1,
            const char hobbies2[MAX_HOBBIE_LENGTH][MAX_HOBBIE_LENGTH],
            int count2,
            int age1,
            int age2,
            const char * personality1,
            const char * personality2)
```

Calcula la similitud de Jaccard entre dos conjuntos de hobbies. Esta función calcula la similitud de Jaccard entre los hobbies de dos usuarios, tomando en cuenta el número de hobbies comunes y el número total de hobbies.

Parameters

hobbies1	Hobbies del primer usuario.
hobbies1_count	Número de hobbies del primer usuario.

hobbies2	Hobbies del segundo usuario.
hobbies2_count	Número de hobbies del segundo usuario.
common_hobbies_count	Número de hobbies comunes entre los dos usuarios.
total_hobbies_count	Número total de hobbies combinados de ambos usuarios.
personality1	Personalidad del primer usuario.
personality2	Personalidad del segundo usuario.

Returns

Valor numérico que representa la similitud de Jaccard entre los hobbies de ambos usuarios.

Calcula la similitud de Jaccard entre dos conjuntos de hobbies. Esta función calcula la similitud de Jaccard entre los hobbies de dos usuarios, tomando en cuenta el número de hobbies comunes y el número total de hobbies.

Parameters

hobbies1	Conjunto de hobbies del primer usuario.
count1	Número de hobbies del primer usuario.
hobbies2	Conjunto de hobbies del segundo usuario.
count2	Número de hobbies del segundo usuario.
age1	Edad del primer usuario.
age2	Edad del segundo usuario.
personality1	Personalidad del primer usuario.
personality2	Personalidad del segundo usuario.

Returns

Un valor entre 0 y 1 que representa la similitud entre los dos usuarios.

Calcula el índice de similitud de Jaccard entre dos conjuntos de hobbies.

```
int intersection = 0, union_count = 0;
char seen[MAX_HOBBIES][MAX_HOBBIE_LENGTH] = {0};
int seen_count = 0;
for (int i = 0; i < count1; i++)
  for (int j = 0; j < count2; j++)
      contar interseccion
for (int i = 0; i < count1; i++)
for (int j = 0; j < count2; j++)
  contar union
union_count = seen_count;
double jaccard = union_count > 0 ? (double)intersection / union_count : 0.0;
double age_weight = calculate_age_weight(age1, age2);
int group1 = get_personality_group(personality1);
int group2 = get_personality_group(personality2);
double personality_multiplier = calculate_personality_multiplier(group1, group2);
return jaccard * age_weight * personality_multiplier; -> Retorna el puntaje ajustado por edad y el
      multiplicador de personalidad
```

Definition at line 22 of file similarity.c.

5.9.2.3 calculate_personality_multiplier()

```
double calculate_personality_multiplier (
            int group1,
            int group2)
```

Calcula un multiplicador de personalidad entre dos usuarios. Compara las personalidades de dos usuarios y devuelve un multiplicador que indica cuán compatibles son sus personalidades.

Parameters

personality1	Personalidad del primer usuario.
personality2	Personalidad del segundo usuario.

Returns

Un valor numérico que representa la compatibilidad de personalidad.

Calcula un multiplicador de personalidad entre dos usuarios. Compara las personalidades de dos usuarios y devuelve un multiplicador que indica cuán compatibles son sus personalidades.

Parameters

group1	El grupo de personalidad del primer usuario.
group2	El grupo de personalidad del segundo usuario.

Returns

Un valor flotante que representa el multiplicador de compatibilidad de personalidad.

Calcula el multiplicador de personalidad entre dos usuarios.Si ambos usuarios pertenecen al mismo grupo de personalidad, se aumenta el multiplicador. Si no, se reduce.

```
double personality_multiplier = 1.0;
if (group1 == group2)
   personality_multiplier = 1.2;
else if (group1 == 0 || group2 == 0)
  personality_multiplier = 1.0;
else
  personality_multiplier = 0.8;
return personality_multiplier;
```

Definition at line 82 of file parameters.c.

5.9.2.4 explain_personality_compatibility()

```
void explain_personality_compatibility (
             const User * user1,
             const User * user2)
```

Explica la compatibilidad de personalidad entre dos usuarios. Imprime una descripción detallada de cómo las personalidades de dos usuarios son compatibles.

Parameters

user1	Puntero al primer usuario.
user2	Puntero al segundo usuario.

Explica la compatibilidad de personalidad entre dos usuarios. Imprime una descripción detallada de cómo las personalidades de dos usuarios son compatibles.

Parameters

user1	El primer usuario.
user2	El segundo usuario.

Explica la compatibilidad de personalidad entre dos usuarios,Compara los grupos de personalidad de dos usuarios e imprime su nivel de compatibilidad.

```
const char *group_names[] = {"Sin grupo", "Analistas (Racionales)", "Diplomáticos (Idealistas)", "Centinelas
      (Conservadores)", "Exploradores (Artísticos)"};
int group1 = get_personality_group(user1->personality);
int group2 = get_personality_group(user2->personality);
if (group1 == 0 || group2 == 0)
  no tiene compatibilidad
if (group1 == group2)
  compatibilidad alta
else
  compatibilidad baja
```

Definition at line 43 of file parameters.c.

5.9.2.5 find_common_hobbies()

```
void find_common_hobbies (
            const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            int count1,
            const char hobbies2[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            int count2)
```

Encuentra los hobbies comunes entre dos usuarios. Esta función compara los hobbies de dos usuarios y encuentra aquellos que coinciden.

Parameters

hobbies1	Hobbies del primer usuario.
hobbies1_count	Número de hobbies del primer usuario.
hobbies2	Hobbies del segundo usuario.
hobbies2_count	Número de hobbies del segundo usuario.

Encuentra los hobbies comunes entre dos usuarios. Esta función compara los hobbies de dos usuarios y encuentra aquellos que coinciden.

Parameters

hobbies1	Lista de hobbies del primer usuario.
count1	El número de hobbies del primer usuario.
hobbies2	Lista de hobbies del segundo usuario.
count2	El número de hobbies del segundo usuario.

Encuentra los hobbies comunes entre dos usuarios, compara los hobbies de ambos usuarios e imprime los que son comunes

```
fprintf(stdout, "   - Hobbies en común: ");
int found_common = 0;
char seen[MAX_HOBBIES][MAX_HOBBIE_LENGTH];
int seen_count = 0;
for (int i = 0; i < count1; i++)
  Almacenar los hobbies del primer usuario
for (int j = 0; j < count2; j++)
  Verificar los hobbies del segundo usuario
if (!found_common)
   fprintf(stdout, "Ninguno");

fprintf(stdout, "\n");
```

Definition at line 187 of file parameters.c.

5.9.2.6 get_age_compatibility_level()

```
const char * get_age_compatibility_level (
            int age_diff)
```

Obtiene el nivel de compatibilidad de edad entre dos usuarios. Esta función clasifica la compatibilidad de edad entre dos usuarios en categorías.

Parameters

age_diff	Diferencia de edad entre los dos usuarios.

Returns

Una cadena que describe el nivel de compatibilidad de edad.

Obtiene el nivel de compatibilidad de edad entre dos usuarios. Esta función clasifica la compatibilidad de edad entre dos usuarios en categorías.

Parameters

age_diff	La diferencia de edad entre dos usuarios.

Returns

Una cadena de texto que indica el nivel de compatibilidad basado en la diferencia de edad.

Obtiene el nivel de compatibilidad basado en la diferencia de edad.

```
if (age_diff <= 5)
  return "Excelente";
else if (age_diff <= 10)
  return "Buena";
else if (age_diff <= 15)
  return "Moderada";
else if (age_diff <= 20)
  return "Baja";
else
 return "Muy baja";
```

Definition at line 114 of file parameters.c.

5.9.2.7 get_personality_group()

```
int get_personality_group (
            const char * personality)
```

Obtiene el grupo de personalidad al que pertenece un usuario. La personalidad de un usuario se clasifica en un grupo de personalidades predefinido.

Parameters

personality	Personalidad del usuario.

Returns

El grupo al que pertenece la personalidad del usuario.

Obtiene el grupo de personalidad al que pertenece un usuario. La personalidad de un usuario se clasifica en un grupo de personalidades predefinido.

Parameters

personality	La personalidad del usuario.

Returns

El grupo de personalidad correspondiente (1: Analistas, 2: Diplomáticos, 3: Sentinelas, 4: Exploradores, 0 si no se encuentra).

Determina el grupo de personalidad de un usuario,compara las primeras tres letras de la personalidad del usuario y devuelve un grupo.

```
if (personality == NULL || strlen(personality) < 5) ->error
if (strncmp(personality, "INT", 3) == 0 || strncmp(personality, "ENT", 3) == 0)
  Compara las primeras 3 letras de la personalidad
```

Definition at line 14 of file parameters.c.

5.9.2.8 partition()

```
int partition (
            Match matches[],
            int low,
            int high)
```

Particiona un arreglo de coincidencias para el algoritmo QuickSort. Se utiliza para reorganizar el arreglo y preparar el pivote para la ordenación.

Parameters

matches	Arreglo de coincidencias entre usuarios.
low	Índice bajo del arreglo.
high	Índice alto del arreglo.

Returns

El índice del pivote después de la partición.

Particiona un arreglo de coincidencias para el algoritmo QuickSort. Se utiliza para reorganizar el arreglo y preparar el pivote para la ordenación.

Parameters

matches	El arreglo de matches a ordenar.
low	El índice más bajo del subarreglo.
high	El índice más alto del subarreglo.

Returns

El índice de partición que divide el arreglo en dos subarreglos.

```
double pivot = matches[high].similarity;
int i = low - 1;
for (int j = low; j < high; j++)
   Intercambiar elementos
Match temp = matches[i + 1];
matches[i + 1] = matches[high];
matches[high] = temp;
```

Definition at line 267 of file parameters.c.

5.9.2.9 quicksort()

```
void quicksort (
            Match matches[],
            int low,
            int high)
```

Ordena un arreglo de coincidencias de usuarios utilizando el algoritmo QuickSort. Este algoritmo organiza las coincidencias de acuerdo con su similitud, de mayor a menor.

Parameters

matches	Arreglo de coincidencias entre usuarios.
low	Índice bajo del arreglo.
high	Índice alto del arreglo.

Ordena un arreglo de coincidencias de usuarios utilizando el algoritmo QuickSort. Este algoritmo organiza las coincidencias de acuerdo con su similitud, de mayor a menor.

Parameters

matches	El arreglo de matches a ordenar.
low	El índice más bajo del subarreglo.
high	El índice más alto del subarreglo.

Ordena un arreglo de matches utilizando el algoritmo de quicksort, el arreglo se ordena de mayor a menor según la similitud entre los usuarios.

```
if (low < high)
Ordena un arreglo de matches utilizando el algoritmo de quicksort, el arreglo se ordena de mayor a menor
      según la similitud entre los usuarios.
```

Definition at line 243 of file parameters.c.

5.9.2.10 recommend_users()

```
void recommend_users (
            const User users[MAX_USERS],
            int num_users)
```

Recomienda usuarios basándose en la similitud. Compara todos los usuarios entre sí y recomienda aquellos con mayor compatibilidad basada en la similitud de hobbies, edad y personalidad.

Parameters

users	Arreglo de usuarios en el sistema.
user_count	Número de usuarios en el sistema.

Recomienda usuarios basándose en la similitud. Compara todos los usuarios entre sí y recomienda aquellos con mayor compatibilidad basada en la similitud de hobbies, edad y personalidad.

Parameters

users	Arreglo de usuarios a evaluar.
num_users	Número total de usuarios.

Recomienda usuarios basándose en la similitud de hobbies, edad y personalidad.

```
for (int i = 0; i < num_users; i++)
 fprintf(stdout, CYAN "\nUsuario %d (%s, %d años):" RESET, users[i].id, users[i].username, users[i].age);
 Match matches[MAX_USERS];
 int match_count = 0;
 int count1 = 0, count2 = 0;
 for (int k = 0; k < MAX_HOBBIES && strlen(users[i].hobbies[k]) > 0; k++)
      Calcular el número de hobbies para el usuario i.
 for (int j = 0; j < num_users; j++)
      Encontrar todas las coincidencias.
 if (match_count > 1) -> Ordenar las coincidencias por similitud (método quicksort).
 int show_matches = match_count > 3 ? 3 : match_count;
 if (show_matches > 0) -> Mostrar las mejores coincidencias (hasta 3)
```

Definition at line 94 of file similarity.c.

5.10 similarity.h

Go to the documentation of this file.

```
00001 #ifndef SIMILARITY_H
00002 #define SIMILARITY_H
00003
00012 #include "users.h"
00013
00027 /* Funciones dedicadas a la Similitud */
00028 double calculate_jaccard_similarity(const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int, const
      char hobbies2[MAX_HOBBIE_LENGTH][MAX_HOBBIE_LENGTH], int, int, int, const char *, const char *);
00029
00038 void find_common_hobbies(const char[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int, const
      char[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int);
00039
00046 void recommend_users(const User users[MAX_USERS], int);
00047
00055 double calculate_age_weight(int, int);
00056
00063 const char *get_age_compatibility_level(int);
00064
00072 double calculate_personality_multiplier(int, int);
00073
00080 void explain_personality_compatibility(const User *, const User *);
00081
00088 int get_personality_group(const char *);
00089
00097 void quicksort(Match matches[], int, int);
00098
00107 int partition(Match matches[], int, int);
00108
00109 #endif
```

5.11 incs/users.h File Reference

Definición de estructuras y funciones para manejar usuarios. Este archivo contiene las constantes, estructuras y funciones necesarias para manejar la creación, inicialización y administración de usuarios en el sistema.

Classes

- struct User

  Estructura que representa a un usuario. Esta estructura contiene la información básica de un usuario, como su ID, nombre, edad, género, hobbies y personalidad.
- struct Match

  Estructura que representa una coincidencia entre dos usuarios. Esta estructura se utiliza para almacenar la similitud de hobbies y la diferencia de edad entre dos usuarios que se han emparejado en el sistema.

Macros

- #define MAX_USERS 51
- #define MAX_NAME_LENGTH 50
- #define MAX_GENDER 10
- #define MAX_HOBBIES 10
- #define MAX_HOBBIE_LENGTH 50
- #define MAX_AGE 60
- #define MIN_AGE 18
- #define MAX_PERS_LENGTH 50
- #define NUM_PERSONALITY_TYPES 16
- #define MAX_FILE_LINES 100

Typedefs

- typedef struct User **User**

Functions

- void load_file (const char *, char[MAX_FILE_LINES][MAX_NAME_LENGTH], int *)

  Carga un archivo de texto en un arreglo. Lee un archivo de texto y almacena cada línea en un arreglo bidimensional de cadenas.
- void generate_random_users (User *, int, char[MAX_FILE_LINES][MAX_NAME_LENGTH], int, char[MAX_FILE_LINES][MAX_NAME_LENGTH], int, char[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int, char[MAX_FILE_LINES][MAX_PERS_LENGTH], int)

  Genera un usuario aleatorio. Crea un usuario con información aleatoria, como nombre, edad, género, hobbies y personalidad, seleccionados de listas predefinidas.
- void generate_random_hobbies (char[MAX_HOBBIES][MAX_HOBBIE_LENGTH], char[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int)

  Genera hobbies aleatorios para un usuario. Selecciona aleatoriamente un número de hobbies de una lista predefinida y los asigna al usuario.
- void generate_random_personality (char *, char[MAX_FILE_LINES][MAX_PERS_LENGTH], int)

  Genera una personalidad aleatoria para un usuario. Selecciona aleatoriamente una personalidad de una lista predefinida y la asigna al usuario.
- void print_users (const User *)

  Imprime la información de un usuario. Imprime los detalles de un usuario, incluyendo su ID, nombre, género, edad, personalidad y hobbies en la salida estándar.

5.11.1 Detailed Description

Definición de estructuras y funciones para manejar usuarios. Este archivo contiene las constantes, estructuras y funciones necesarias para manejar la creación, inicialización y administración de usuarios en el sistema.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Definition in file users.h.

5.11.2 Macro Definition Documentation

5.11.2.1 MAX_AGE

```
#define MAX_AGE 60
```

Definition at line 32 of file users.h.

5.11.2.2 MAX_FILE_LINES

```
#define MAX_FILE_LINES 100
```

Definition at line 36 of file users.h.

5.11.2.3 MAX_GENDER

```
#define MAX_GENDER 10
```

Definition at line 29 of file users.h.

5.11.2.4 MAX_HOBBIE_LENGTH

```
#define MAX_HOBBIE_LENGTH 50
```

Definition at line 31 of file users.h.

5.11.2.5 MAX_HOBBIES

```
#define MAX_HOBBIES 10
```

Definition at line 30 of file users.h.

5.11.2.6 MAX_NAME_LENGTH

```
#define MAX_NAME_LENGTH 50
```

Definition at line 28 of file users.h.

5.11.2.7 MAX_PERS_LENGTH

```
#define MAX_PERS_LENGTH 50
```

Definition at line 34 of file users.h.

5.11.2.8 MAX_USERS

```
#define MAX_USERS 51
```

Macros utilizadas en el proyecto.

```
#define MAX_USERS 51
#define MAX_NAME_LENGTH 50
#define MAX_GENDER 10
#define MAX_HOBBIES 10
#define MAX_HOBBIE_LENGTH 50
#define MAX_AGE 60
#define MIN_AGE 18
#define MAX_PERS_LENGTH 50
#define NUM_PERSONALITY_TYPES 16
#define MAX_FILE_LINES 100
```

Definition at line 27 of file users.h.

5.11.2.9 MIN_AGE

```
#define MIN_AGE 18
```

Definition at line 33 of file users.h.

5.11.2.10 NUM_PERSONALITY_TYPES

```
#define NUM_PERSONALITY_TYPES 16
```

Definition at line 35 of file users.h.

5.11.3 Function Documentation

5.11.3.1 generate_random_hobbies()

```
void generate_random_hobbies (
            char hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH],
            int hobby_count)
```

Genera hobbies aleatorios para un usuario. Selecciona aleatoriamente un número de hobbies de una lista predefinida y los asigna al usuario.

Parameters

hobbies	Arreglo donde se almacenarán los hobbies generados.
hobbies_list	Lista de hobbies disponibles para la selección.
hobby_count	Número total de hobbies disponibles en la lista.

Genera hobbies aleatorios para un usuario. Selecciona aleatoriamente un número de hobbies de una lista predefinida y los asigna al usuario.

Parameters

hobbies	Arreglo donde se almacenarán los hobbies generados.
hobbies_list	Lista de hobbies disponibles para selección.
hobby_count	Número total de hobbies disponibles en la lista.

Genera hobbies aleatorios para un usuario.

```
*if (hobby_count <= 0) ->Error
for (int i = 0; i < MAX_HOBBIES; i++)
int *hobbie_selected = calloc(hobby_count, sizeof(int));
if (!hobbie_selected) ->return
int num_hobbies = (random() % MAX_HOBBIES) + 1;
int added_hobbies = 0;
for (int i = 0; i < num_hobbies && added_hobbies < MAX_HOBBIES; i++)
 Selecciona aleatoriamente hobbies únicos de una lista disponible, los asigna al usuario y asegura que no se
      exceda el número máximo permitido.
```

Definition at line 130 of file user.c.

5.11.3.2 generate_random_personality()

```
void generate_random_personality (
            char * personality,
            char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH],
            int personality_count)
```

Genera una personalidad aleatoria para un usuario. Selecciona aleatoriamente una personalidad de una lista predefinida y la asigna al usuario.

Parameters

personality	Puntero a la cadena donde se almacenará la personalidad generada.
personalities_list	Lista de personalidades disponibles para la selección.
personality_count	Número total de personalidades disponibles en la lista.

Genera una personalidad aleatoria para un usuario. Selecciona aleatoriamente una personalidad de una lista predefinida y la asigna al usuario.

Parameters

personality	Puntero a una cadena donde se almacenará la personalidad generada.
personalities_list	Lista de personalidades disponibles para selección.
personality_count	Número total de personalidades disponibles en la lista.

Genera una personalidad aleatoria para un usuario.

```
  if (personality_count <= 0) ->error
  personality[0] = '\0';
*strcpy(personality, personalities_list[random() % personality_count]);
```

Definition at line 186 of file user.c.

5.11.3.3 generate_random_users()

```
void generate_random_users (
            User * user,
            int id,
            char male_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH],
            int male_count,
            char female_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH],
            int female_count,
            char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH],
            int hobby_count,
            char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH],
            int personality_count)
```

Genera un usuario aleatorio. Crea un usuario con información aleatoria, como nombre, edad, género, hobbies y personalidad, seleccionados de listas predefinidas.

Parameters

user	Puntero al usuario a generar.
id	ID único que se asignará al usuario.
male_usernames	Lista de nombres masculinos disponibles.
male_count	Número de nombres masculinos en la lista.
female_usernames	Lista de nombres femeninos disponibles.
female_count	Número de nombres femeninos en la lista.
hobbies_list	Lista de hobbies disponibles.
hobby_count	Número de hobbies en la lista.
personalities_list	Lista de personalidades disponibles.
personality_count	Número de personalidades en la lista.

Genera un usuario aleatorio. Crea un usuario con información aleatoria, como nombre, edad, género, hobbies y personalidad, seleccionados de listas predefinidas.

Parameters

user	Puntero al usuario que se generará.
id	ID único que se asignará al usuario.
male_usernames	Lista de nombres de usuario masculinos disponibles.
male_count	Número de nombres masculinos en la lista.
female_usernames	Lista de nombres de usuario femeninos disponibles.
female_count	Número de nombres femeninos en la lista.
hobbies_list	Lista de hobbies disponibles.
hobby_count	Número de hobbies en la lista.
personalities_list	Lista de personalidades disponibles.
personality_count	Número de personalidades en la lista.

Genera un usuario aleatorio.

```
if (!user) ->Error
user->id = id;
user->age = random() % MAX_AGE + MIN_AGE;
int gender_choice = random() % 2;
if (gender_choice == 0 && male_count > 0)
   Asigna al usuario género "Masculino" y un nombre aleatorio de la lista de nombres masculinos si hay
      nombres disponibles y el género seleccionado es masculino.
```

```
 else if (female_count > 0)
      Asigna al usuario género "Femenino" y un nombre aleatorio de la lista de nombres femeninos si hay
      nombres disponibles.
generate_random_hobbies(user->hobbies, hobbies_list, hobby_count);
generate_random_personality(user->personality, personalities_list, personality_count);
```

Definition at line 70 of file user.c.

5.11.3.4 load_file()

```
void load_file (
            const char * filename,
            char file_array[MAX_FILE_LINES][MAX_NAME_LENGTH],
            int * count)
```

Carga un archivo de texto en un arreglo. Lee un archivo de texto y almacena cada línea en un arreglo bidimensional de cadenas.

Parameters

filename	Nombre del archivo a cargar.
file_array	Arreglo donde se almacenarán las líneas leídas del archivo.
count	Puntero a la variable donde se almacenará el número de líneas leídas.

Carga un archivo de texto en un arreglo. Lee un archivo de texto y almacena cada línea en un arreglo bidimensional de cadenas.

Parameters

filename	Nombre del archivo a cargar.
file_array	Arreglo donde se almacenarán las líneas del archivo.
count	Puntero a la variable que almacena el número de líneas leídas.

Carga un archivo de texto en un arreglo.

```
 Nombre del archivo a cargar.
FILE *file = fopen(filename, "r");
if (!file) ->Error
*count = 0;
char line[MAX_NAME_LENGTH];
while (*count < MAX_FILE_LINES && fgets(line, sizeof(line), file))
    Lee líneas del archivo y las guarda en un arreglo hasta alcanzar el límite o el final del archivo.
fclose(file);
```

Definition at line 16 of file user.c.

5.11.3.5 print_users()

```
void print_users (
            const User * user)
```

Imprime la información de un usuario. Imprime los detalles de un usuario, incluyendo su ID, nombre, género, edad, personalidad y hobbies en la salida estándar.

Parameters

user	Puntero al usuario cuya información se desea imprimir.

Imprime la información de un usuario. Imprime los detalles de un usuario, incluyendo su ID, nombre, género, edad, personalidad y hobbies en la salida estándar.

Parameters

user	Puntero al usuario que se desea imprimir.

Imprime la información de un usuario.

```
if (!user) ->Error
fprintf(stdout, "ID: %d\n", user->id);
fprintf(stdout, "Nombre: %s\n", user->username);
fprintf(stdout, "Género: %s\n", user->gender);
fprintf(stdout, "Edad: %d\n", user->age);
fprintf(stdout, "Personalidad: %s\n", user->personality);
fprintf(stdout, "Hobbies:\n");
for (int i = 0; i < MAX_HOBBIES && user->hobbies[i][0] != '\0'; i++)
  Muestra el hobbie del usuario.
```

Definition at line 214 of file user.c.

5.12 users.h

Go to the documentation of this file.

```
00001 #ifndef USERS_H
00002 #define USERS_H
00003
00027 #define MAX_USERS 51
00028 #define MAX_NAME_LENGTH 50
00029 #define MAX_GENDER 10
00030 #define MAX_HOBBIES 10
00031 #define MAX_HOBBIE_LENGTH 50
00032 #define MAX_AGE 60
00033 #define MIN_AGE 18
00034 #define MAX_PERS_LENGTH 50
00035 #define NUM_PERSONALITY_TYPES 16
00036 #define MAX_FILE_LINES 100
00037
00055 /* Estructura del Usuario */
00056 typedef struct User
00057 {
00058     int id;
00059     char username[MAX_NAME_LENGTH];
00060     int age;
00061     char gender[MAX_GENDER];
00062     char hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH];
00063     char personality[MAX_PERS_LENGTH];
00064 } User;
00065
00080 typedef struct
00081 {
00082     int user_index;
00083     double similarity;
00084     int age_diff;
00085 } Match;
00086
00094 void load_file(const char *, char[MAX_FILE_LINES][MAX_NAME_LENGTH], int *);
00095
00110 void generate_random_users(User *, int, char[MAX_FILE_LINES][MAX_NAME_LENGTH], int,
      char[MAX_FILE_LINES][MAX_NAME_LENGTH], int, char[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int,
      char[MAX_FILE_LINES][MAX_PERS_LENGTH], int);
00111
00119 void generate_random_hobbies(char[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
      char[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int);
00120
00128 void generate_random_personality(char *, char[MAX_FILE_LINES][MAX_PERS_LENGTH], int);
00129
00135 void print_users(const User *);
00136
00137 #endif
```

5.13 src/connections_graph.c File Reference

Funciones para la gestión y visualización de grafos de conexiones entre usuarios.

```
#include "main.h"
```

Functions

- Graph * initialize_graph (int numUsers, User *users)
  Inicializa un grafo con un número específico de usuarios y sus nombres.
- void add_connection (Graph *graph, int user1, int user2)
  Agrega una conexión bidireccional entre dos usuarios en el grafo.
- void display_graph (Graph *graph, int source)
  Muestra las conexiones del grafo y el camino más largo desde un nodo fuente utilizando Dijkstra.
- void print_path (int target, int *prev_node, Graph *graph)
  Muestra el camino desde un nodo fuente hasta un objetivo.
- void free_graph (Graph *graph)
  Libera toda la memoria asignada al grafo.

5.13.1 Detailed Description

Funciones para la gestión y visualización de grafos de conexiones entre usuarios.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras. Benjamin Sanhueza y Johann Fink-

Definition in file connections_graph.c.

5.13.2 Function Documentation

5.13.2.1 add_connection()

```
void add_connection (
            Graph * graph,
            int user1,
            int user2)
```

Agrega una conexión bidireccional entre dos usuarios en el grafo.

Agrega una conexión bidireccional entre dos usuarios en el grafo. Inserta nodos en la lista de adyacencia para conectar a dos usuarios,representando una relación de amistad.

Parameters

graph	Puntero al grafo donde se agregarán las conexiones.
user1	Identificador del primer usuario.
user2	Identificador del segundo usuario.

Agrega una conexión bidireccional entre dos usuarios en el grafo.

```
if (!graph) ->error
Node *newNode = (Node *)malloc(sizeof(Node));
newNode->id = user2;
newNode->weight = 1;
newNode->next = graph->adjacencyList[user1];
graph->adjacencyList[user1] = newNode;
newNode = (Node *)malloc(sizeof(Node));
newNode->id = user1;
newNode->weight = 1;
newNode->next = graph->adjacencyList[user2];
graph->adjacencyList[user2] = newNode;
```

Definition at line 55 of file connections_graph.c.

5.13.2.2 display_graph()

```
void display_graph (
            Graph * graph,
            int source)
```

Muestra las conexiones del grafo y el camino más largo desde un nodo fuente utilizando Dijkstra.

Muestra las conexiones del grafo utilizando el algoritmo de Dijkstra. Calcula las rutas más cortas desde un usuario específico al resto utilizando Dijkstra y muestra la distancia y el camino hacia el nodo más lejano alcanzable.

Parameters

graph	Puntero al grafo cuyas conexiones se mostrarán.
source	Identificador del nodo fuente desde el cual calcular las distancias.

Muestra las conexiones del grafo y el camino más largo desde un nodo fuente utilizando Dijkstra.

```
double distance[MAX_USERS];
int prev_node[MAX_USERS];
int visited_node[MAX_USERS] = {0};
for(int i=0;i<graph->numUsers;i++)
  Aqui se inicializan las distancias de los nodos desde el nodo fuente y cual es el nodo previo
distance[source]=0;
for(int i=0;i<graph->numUsers;i++)
   Algoritmo Dijkstra
int farthest_node=-1;
int max_distance=-1;
```

Definition at line 96 of file connections_graph.c.

5.13.2.3 free_graph()

```
void free_graph (
            Graph * graph)
```

Libera toda la memoria asignada al grafo.

Libera toda la memoria asociada con el grafo. Elimina todas las estructuras dinámicas asociadas con la lista de adyacencia y los nombres de los usuarios.

Parameters

graph	Puntero al grafo a liberar.

Libera toda la memoria asignada al grafo

```
for (int i = 0; i < graph->numUsers; i++)
  libera nombres de usuarios
free(graph->user_names);
free(graph->adjacencyList);
free(graph);
```

Definition at line 205 of file connections_graph.c.

5.13.2.4 initialize_graph()

```
Graph * initialize_graph (
            int numUsers,
            User * users)
```

Inicializa un grafo con un número específico de usuarios y sus nombres.

Inicializa un grafo con un número específico de usuarios y sus nombres. Reserva memoria y configura una estructura de grafo, inicializando la lista de adyacencia y copiando los nombres de los usuarios.

Parameters

numUsers	Número total de usuarios en el grafo.
users	Puntero al arreglo de usuarios, cada uno con su nombre.

Returns

Puntero al grafo inicializado.

Inicializa un grafo con un número específico de usuarios y sus nombres.

```
if (numUsers <= 0) ->error
Graph *graph = (Graph *)malloc(sizeof(Graph));
graph->numUsers = numUsers;
graph->adjacencyList = (Node **)calloc(numUsers + 1, sizeof(Node *));
graph->user_names = malloc(numUsers * sizeof(char *));
for (int i = 0; i < numUsers; i++)
  Agregar nombres de usuarios.
return graph;
```

Definition at line 15 of file connections_graph.c.

5.13.2.5 print_path()

```
void print_path (
            int target,
            int * prev_node,
            Graph * graph)
```

Muestra el camino desde un nodo fuente hasta un objetivo.

Imprime el camino desde un nodo fuente a un nodo destino. Muestra la secuencia de nombres de usuarios que forman la ruta más corta calculada previamente.

Parameters

target	Nodo objetivo.
prev_node	Arreglo con los predecesores de cada nodo.
graph	Puntero al grafo.

Muestra el camino desde un nodo fuente hasta un objetivo.

```
if (prev_node[target] == -1)
  fprintf(stdout, "%s", graph->user_names[target]);
  return;
print_path(prev_node[target], prev_node, graph);
fprintf(stdout, " -> %s", graph->user_names[target]);
```

Definition at line 181 of file connections_graph.c.

5.14 connections_graph.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00015 Graph *initialize_graph(int numUsers, User *users)
00016 {
00030     if (numUsers <= 0)
00031     {
00032         fprintf(stderr, "No se puede inicializar un grafo sin usuarios. Saliendo...\n");
00033         exit(EXIT_FAILURE);
00034     }
00035
00036     Graph *graph = (Graph *)malloc(sizeof(Graph));
00037
00038     graph->numUsers = numUsers;
00039
00040     graph->adjacencyList = (Node **)calloc(numUsers + 1, sizeof(Node *));
00041
00042     graph->user_names = malloc(numUsers * sizeof(char *));
00043     for (int i = 0; i < numUsers; i++)
00044         graph->user_names[i] = strdup(users[i].username);
00045
00046     return graph;
00047 }
00048
00055 void add_connection(Graph *graph, int user1, int user2)
00056 {
00073     if (!graph)
00074     {
00075         fprintf(stderr, "Grafo no inicializado. Saliendo...\n");
00076         exit(EXIT_FAILURE);
00077     }
00078
00079     Node *newNode = (Node *)malloc(sizeof(Node));
00080     newNode->id = user2;
00081     newNode->weight = 1;
00082     newNode->next = graph->adjacencyList[user1];
00083     graph->adjacencyList[user1] = newNode;
00084
00085     newNode = (Node *)malloc(sizeof(Node));
00086     newNode->id = user1;
00087     newNode->weight = 1;
00088     newNode->next = graph->adjacencyList[user2];
00089     graph->adjacencyList[user2] = newNode;
00090 }
00096 void display_graph(Graph *graph, int source)
00097 {
00113     double distance[MAX_USERS];
00114     int prev_node[MAX_USERS];
00115     int visited_node[MAX_USERS] = {0};
00116
00117     for (int i = 0; i < graph->numUsers; i++)
00118     {
00119         distance[i] = INFINITY;
00120         prev_node[i] = -1;
00121     }
00122     distance[source] = 0;
00123
```

```
00124     for (int i = 0; i < graph->numUsers; i++)
00125     {
00126         int min_index = -1;
00127         double min_distance = INFINITY;
00128 
00129         for (int i = 0; i < graph->numUsers; i++)
00130             if (!visited_node[i] && distance[i] < min_distance)
00131             {
00132                 min_distance = distance[i];
00133                 min_index = i;
00134             }
00135         if (min_index == -1)
00136             break;
00137         visited_node[min_index] = 1;
00138         Node *current = graph->adjacencyList[min_index];
00139 
00140         while (current)
00141         {
00142             int neighbor = current->id;
00143             int weight = current->weight;
00144 
00145             if (!visited_node[neighbor] && distance[min_index] + weight < distance[neighbor])
00146             {
00147                 distance[neighbor] = distance[min_index] + weight;
00148                 prev_node[neighbor] = min_index;
00149             }
00150 
00151             current = current->next;
00152         }
00153     }
00154 
00155     int farthest_node = -1;
00156     int max_distance = -1;
00157 
00158     for (int i = 0; i < graph->numUsers; i++)
00159         if (distance[i] != INFINITY && distance[i] > max_distance)
00160         {
00161             max_distance = distance[i];
00162             farthest_node = i;
00163         }
00164 
00165     if (farthest_node != -1)
00166     {
00167         fprintf(stdout, GREEN "%-10s: " RESET, graph->user_names[source]);
00168         fprintf(stdout, "Distancia: %d , Camino: ", max_distance);
00169         print_path(farthest_node, prev_node, graph);
00170         fprintf(stdout, "\n");
00171     }
00172     else
00173         fprintf(stdout, RED "No se encontraron caminos desde '%s'.\n" RESET,
    graph->user_names[source]);
00174 }
00181 void print_path(int target, int *prev_node, Graph *graph)
00182 {
00193     if (prev_node[target] == -1)
00194     {
00195         fprintf(stdout, "%s", graph->user_names[target]);
00196         return;
00197     }
00198     print_path(prev_node[target], prev_node, graph);
00199     fprintf(stdout, " -> %s", graph->user_names[target]);
00200 }
00205 void free_graph(Graph *graph)
00206 {
00217     for (int i = 0; i < graph->numUsers; i++)
00218     {
00219         Node *temp = graph->adjacencyList[i];
00220         while (temp)
00221         {
00222             Node *toDelete = temp;
00223             temp = temp->next;
00224             free(toDelete);
00225         }
00226 
00227         free(graph->user_names[i]);
00228     }
00229 
00230     free(graph->user_names);
00231     free(graph->adjacencyList);
00232     free(graph);
00233 }
```

5.15 src/graphic.c File Reference

funciones para dibujar la conexion entre los grafos

```
#include "main.h"
```

Functions

- void generate_eps_graph (Graph *graph, const char *filename)

  Genera un archivo EPS que representa un grafo y lo convierte a PNG.
- void transform_eps_png (const char *filename)

  Convierte un archivo EPS a PNG y elimina el archivo EPS. Utiliza Ghostscript para realizar la conversión y asegura que el archivo EPS se elimine después de completar el proceso.

5.15.1 Detailed Description

funciones para dibujar la conexion entre los grafos

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras. Benjamin Sanhueza y Johann Fink

Definition in file graphic.c.

5.15.2 Function Documentation

5.15.2.1 generate_eps_graph()

```
void generate_eps_graph (
            Graph * graph,
            const char * filename)
```

Genera un archivo EPS que representa un grafo y lo convierte a PNG.

Genera un archivo .dot con la representación del grafo.

Parameters

graph	Puntero al grafo que se desea visualizar.
filename	Nombre del archivo EPS a generar.

Note

Si el grafo o el nombre del archivo no son válidos, el programa finaliza con error.

Genera un archivo EPS que representa un grafo y lo convierte a PNG.

```
if (!graph || !filename) ->error
FILE *file = fopen(filename, "w");
if (!file) ->error
int radius = 200;
int centerX = 250;
int centerY = 250;
double angleStep = 2 * M_PI / graph->numUsers;
int positions[MAX_USERS][2];
for (int i = 0; i < graph->numUsers; i++)
  Almacena las posiciones de cada nodo.
for (int i = 0; i < graph->numUsers; i++)
  Dibujar nodos (usuarios) y nombres de cada uno.
transform_eps_png(filename);
```

Definition at line 15 of file graphic.c.

5.15.2.2 transform_eps_png()

```
void transform_eps_png (
            const char * filename)
```

Convierte un archivo EPS a PNG y elimina el archivo EPS. Utiliza Ghostscript para realizar la conversión y asegura que el archivo EPS se elimine después de completar el proceso.

Transforma un archivo .eps a .png.

Parameters

filename	Nombre del archivo EPS a convertir.

Note

Si el nombre del archivo no es válido o la conversión falla, el programa finaliza con error.

Convierte un archivo EPS a PNG y elimina el archivo EPS.

```
if (!filename) ->error
char base_filename[256];
strncpy(base_filename, filename, sizeof(base_filename) - 1);
base_filename[sizeof(base_filename) - 1] = '\0';
char *ext = strrchr(base_filename, '.');
if (ext && strcmp(ext, ".eps") == 0)
  Eliminar la extensión .eps
 char command[512];
snprintf(command, sizeof(command), "gs -dSAFER -dBATCH -dNOPAUSE -dEPSCrop -sDEVICE=png16m -r300
      -sOutputFile=%s.png %s > /dev/null 2>&1", base_filename, filename);
int result = system(command);
if (result != 0) ->error
if (remove(filename) != 0)
  Eliminar el archivo EPS.
```

Definition at line 110 of file graphic.c.

5.16 graphic.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00015 void generate_eps_graph(Graph *graph, const char *filename)
00016 {
00035     if (!graph || !filename)
00036     {
00037         fprintf(stderr, "Grafo o Nombre de archivo no válidos. Saliendo...\n");
00038         exit(EXIT_FAILURE);
00039     }
00040
00041     FILE *file = fopen(filename, "w");
00042     if (!file)
00043     {
00044         fprintf(stderr, "No se pudo crear el archivo %s. Saliendo...\n", filename);
00045         exit(EXIT_FAILURE);
00046     }
00047
00048     fprintf(file, "%%!PS-Adobe-3.0 EPSF-3.0\n");
00049     fprintf(file, "%%%%BoundingBox: 0 0 500 500\n");
00050     fprintf(file, "/circle { newpath 0 360 arc closepath fill } def\n");
00051
00052     int radius = 200;
00053     int centerX = 250;
00054     int centerY = 250;
00055     double angleStep = 2 * M_PI / graph->numUsers;
00056
00057     int positions[MAX_USERS][2];
00058     for (int i = 0; i < graph->numUsers; i++)
```

```
00059     {
00060         double angle = i * angleStep;
00061         positions[i][0] = centerX + radius * cos(angle);
00062         positions[i][1] = centerY + radius * sin(angle);
00063     }
00064 
00065     fprintf(file, "0.8 setgray\n");
00066     for (int i = 0; i < graph->numUsers; i++)
00067     {
00068         Node *current = graph->adjacencyList[i];
00069         while (current)
00070         {
00071             int target = current->id;
00072 
00073             if (i < target)
00074                 fprintf(file, "newpath %d %d moveto %d %d lineto stroke\n", positions[i][0],
      positions[i][1], positions[target][0], positions[target][1]);
00075 
00076             current = current->next;
00077         }
00078     }
00079 
00080     for (int i = 0; i < graph->numUsers; i++)
00081     {
00082         double red = (random() % 128 + 127) / 255.0;
00083         double green = (random() % 128 + 127) / 255.0;
00084         double blue = (random() % 128 + 127) / 255.0;
00085         fprintf(file, "%f %f %f setrgbcolor\n", red, green, blue);
00086         fprintf(file, "%d %d 10 circle\n", positions[i][0], positions[i][1]);
00087 
00088         fprintf(file, "0 setgray\n");
00089         fprintf(file, "/Courier findfont 10 scalefont setfont\n");
00090 
00091         fprintf(file, "newpath %d %d moveto (%s) show\n",
00092                 positions[i][0] - 20, positions[i][1] - 15, graph->user_names[i]);
00093     }
00094 
00095     fclose(file);
00096 
00097     fprintf(stdout, "\nLa imagen del grafo ha sido guardado en la ruta:");
00098     fprintf(stdout, RED " ./output/social_network.png.\n\n" RESET);
00099 
00100     transform_eps_png(filename);
00101 }
00102 
00110 void transform_eps_png(const char *filename)
00111 {
00130     if (!filename)
00131     {
00132         fprintf(stderr, "Nombre de archivo no válido. Saliendo...\n");
00133         exit(EXIT_FAILURE);
00134     }
00135 
00136     char base_filename[256];
00137     strncpy(base_filename, filename, sizeof(base_filename) - 1);
00138     base_filename[sizeof(base_filename) - 1] = '\0';
00139 
00140     char *ext = strrchr(base_filename, '.');
00141     if (ext && strcmp(ext, ".eps") == 0)
00142         *ext = '\0';
00143 
00144     char command[512];
00145     snprintf(command, sizeof(command), "gs -dSAFER -dBATCH -dNOPAUSE -dEPSCrop -sDEVICE=png16m -r300
      -sOutputFile=%s.png %s > /dev/null 2>&1", base_filename, filename);
00146 
00147     int result = system(command);
00148     if (result != 0)
00149     {
00150         fprintf(stderr, "Error al intentar convertir el archivo %s a PNG. Saliendo...\n", filename);
00151         exit(EXIT_FAILURE);
00152     }
00153 
00154     if (remove(filename) != 0)
00155     {
00156         fprintf(stderr, "Error al intentar eliminar el archivo %s. Saliendo...\n", filename);
00157         exit(EXIT_FAILURE);
00158     }
00159 }
```

5.17 src/main.c File Reference

Programa principal que gestiona la creación de usuarios, el grafo de conexiones sociales, y las recomendaciones entre usuarios en base a similitudes, También se encarga de cargar datos desde archivos y generar publicaciones aleatorias.

```
#include "main.h"
```

Functions

- int main (int argc, char ∗argv[ ])

  Función principal que gestiona el flujo del programa, incluyendo la inicialización de usuarios, carga de datos, creación del grafo de conexiones y recomendaciones entre usuarios.

5.17.1 Detailed Description

Programa principal que gestiona la creación de usuarios, el grafo de conexiones sociales, y las recomendaciones entre usuarios en base a similitudes, También se encarga de cargar datos desde archivos y generar publicaciones aleatorias.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamin Sanhueza y Johann Fink-

Definition in file main.c.

5.17.2 Function Documentation

5.17.2.1 main()

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

Función principal que gestiona el flujo del programa, incluyendo la inicialización de usuarios, carga de datos, creación del grafo de conexiones y recomendaciones entre usuarios.

Parameters

argc	Número de argumentos de la línea de comandos.
argv	Arreglo de argumentos pasados al programa.

Returns

EXIT_SUCCESS si la ejecución fue exitosa, EXIT_FAILURE en caso de error.

Función principal que gestiona el flujo del programa, incluyendo la inicialización de usuarios, carga de datos, creación del grafo de conexiones y recomendaciones entre usuarios.

```
int opt;
int num_users = 0;
int exists_users = 0;
int total_users = 0;
while ((opt = getopt(argc, argv, "hu:")) != -1)

if (num_users <= 0 || num_users > MAX_USERS)
  Validar que el número de usuarios sea un valor positivo y menor o igual a 50.
srandom((unsigned int)time(NULL));
Post_List post_list;
init_post_list(&post_list);
char male_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH];
char female_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH];
char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH];
char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH];
int male_count = 0;
int female_count = 0;
int hobby_count = 0;
int personality_count = 0;
load_file("./input/male_usernames.txt", male_usernames, &male_count);
load_file("./input/female_usernames.txt", female_usernames, &female_count);
load_file("./input/hobbies.txt", hobbies_list, &hobby_count);
load_file("./input/personalities.txt", personalities_list, &personality_count);
User users[MAX_USERS]; // Arreglo de usuarios.
if (log_check())
  Verifica la existencia de un historial.
if (total_users > MAX_USERS)
  Capacidad maxima de usuarios alcanzada.
for (int i = exists_users; i < total_users; i++)
  Generar usuarios aleatorios.
log_clean();
fprintf(stdout, RED "\nUsuarios generados:\n" RESET);
for (int i = 0; i < total_users; i++)
  imprime usuarios generados
Graph *socialNetwork = initialize_graph(total_users, users);
if (!socialNetwork)
  Inicializaz el grafo con el numero de usuarios.
double threshold = 0.3;
fprintf(stdout, RED "\nRecomendaciones para los Usuarios:\n" RESET);
recommend_users(users, total_users);
fprintf(stdout, RED "\nConexiones creadas:\n" RESET);
create_connections(users, total_users, socialNetwork, threshold);
fprintf(stdout, RED "\nGrafo de Conexiones:\n\n" RESET);
for (int i = 0; i < total_users; i++)
   Mostrar el grafo.
generate_random_posts(users, total_users, &post_list);
display_all_posts(&post_list);
fprintf(stdout, RED "Usuario con más amigos:\n" RESET);
int userIndex = find_user_with_most_friends(socialNetwork);
if (userIndex != -1)
   print_friends_of_user(socialNetwork, userIndex);
generate_eps_graph(socialNetwork, "./output/social_network.eps");
free_graph(socialNetwork);
free_all_posts(&post_list);
if (total_users >= (MAX_USERS - 10)) -> Warning de cantidad maxima de usuarios cercana.
return EXIT_SUCCESS;
```

Definition at line 17 of file main.c.

5.18 main.c

Go to the documentation of this file.

```
00001
00008 #include "main.h"
00009
00017 int main(int argc, char *argv[])
00018 {
00081     int opt;
00082     int num_users = 0;
00083     int exists_users = 0;
00084     int total_users = 0;
00085
```

```
00086     while ((opt = getopt(argc, argv, "hu:")) != -1)
00087     {
00088         switch (opt)
00089         {
00090         case 'h':
00091             fprintf(stdout, "\nUso del programa:\n");
00092             fprintf(stdout, "  -u <numero_de_usuarios> : Número de usuarios a crear.\n");
00093             fprintf(stdout, "  -h : Muestra esta ayuda.\n\n");
00094             exit(EXIT_SUCCESS);
00095         case 'u':
00096             num_users = atoi(optarg);
00097             break;
00098         case '?':
00099             fprintf(stderr, "Opción no reconocida: -%c\n", optopt);
00100             exit(EXIT_FAILURE);
00101         default:
00102             fprintf(stderr, "Uso: %s [-u numero_de_usuarios]\n", argv[0]);
00103             exit(EXIT_FAILURE);
00104         }
00105     }
00106
00107     if (num_users <= 0 || num_users > MAX_USERS)
00108     {
00109         fprintf(stderr, "Error se debe especificar un valor para -u. Saliendo...\n");
00110         fprintf(stderr, "Uso: %s [-u numero_de_usuarios]\n", argv[0]);
00111         fprintf(stderr, "Para más información, ejecute: %s -h\n", argv[0]);
00112         exit(EXIT_FAILURE);
00113     }
00114
00115     srandom((unsigned int)time(NULL));
00116
00117     Post_List post_list;
00118     init_post_list(&post_list);
00119
00120     char male_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH];
00121     char female_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH];
00122     char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH];
00123     char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH];
00124     int male_count = 0;
00125     int female_count = 0;
00126     int hobby_count = 0;
00127     int personality_count = 0;
00128
00129     load_file("./input/male_usernames.txt", male_usernames, &male_count);
00130     load_file("./input/female_usernames.txt", female_usernames, &female_count);
00131     load_file("./input/hobbies.txt", hobbies_list, &hobby_count);
00132     load_file("./input/personalities.txt", personalities_list, &personality_count);
00133
00134     User users[MAX_USERS];
00135
00136     if (log_check())
00137     {
00138         user_count_from_log(&exists_users);
00139         log_input(users);
00140         total_users = num_users + exists_users;
00141     }
00142     else
00143         total_users = num_users;
00144
00145     if (total_users > MAX_USERS)
00146     {
00147         fprintf(stderr, RED "\nCapacidad de maxima de usuarios alcanzada. Saliendo...\n\n" RESET);
00148         exit(EXIT_FAILURE);
00149     }
00150
00151     for (int i = exists_users; i < total_users; i++)
00152         generate_random_users(&users[i], i + 1, male_usernames, male_count, female_usernames,
      female_count, hobbies_list, hobby_count, personalities_list, personality_count);
00153
00154     log_clean();
00155
00156     fprintf(stdout, RED "\nUsuarios generados:\n" RESET);
00157     for (int i = 0; i < total_users; i++)
00158     {
00159         fprintf(stdout, GREEN "\nUsuario %d:\n" RESET, i + 1);
00160         log_output(&users[i]);
00161         print_users(&users[i]);
00162     }
00163
00164     Graph *socialNetwork = initialize_graph(total_users, users);
00165     if (!socialNetwork)
00166     {
00167         fprintf(stderr, "Error al intentar inicializar el grafo. Saliendo...\n");
00168         exit(EXIT_FAILURE);
00169     }
00170
00171     double threshold = 0.3;
```

```
00172
00173     fprintf(stdout, RED "\nRecomendaciones para los Usuarios:\n" RESET);
00174     recommend_users(users, total_users);
00175
00176     fprintf(stdout, RED "\nConexiones creadas:\n" RESET);
00177     create_connections(users, total_users, socialNetwork, threshold);
00178
00179     fprintf(stdout, RED "\nGrafo de Conexiones:\n\n" RESET);
00180     for (int i = 0; i < total_users; i++)
00181         display_graph(socialNetwork, i);
00182
00183     generate_random_posts(users, total_users, &post_list);
00184
00185     display_all_posts(&post_list);
00186
00187     fprintf(stdout, RED "Usuario con más amigos:\n" RESET);
00188     int userIndex = find_user_with_most_friends(socialNetwork);
00189     if (userIndex != -1)
00190         print_friends_of_user(socialNetwork, userIndex);
00191
00192     generate_eps_graph(socialNetwork, "./output/social_network.eps");
00193
00194     free_graph(socialNetwork);
00195     free_all_posts(&post_list);
00196
00197     if (total_users >= (MAX_USERS - 10))
00198         fprintf(stdout, YELLOW "Se esta alcanzando la capacidad máxima de usuarios, %d Usuarios
      Existentes \n\n" RESET, total_users);
00199
00200     return EXIT_SUCCESS;
00201 }
```

5.19 src/parameters.c File Reference

Funciones para la comparación y recomendación de usuarios basadas en la personalidad, edad y hobbies.

```
#include "main.h"
```

Functions

- int get_personality_group (const char *personality)

  Determina el grupo de personalidad de un usuario,compara las primeras tres letras de la personalidad del usuario y devuelve un grupo.
- void explain_personality_compatibility (const User *user1, const User *user2)

  Explica la compatibilidad de personalidad entre dos usuarios,Compara los grupos de personalidad de dos usuarios e imprime su nivel de compatibilidad.
- double calculate_personality_multiplier (int group1, int group2)

  Calcula el multiplicador de personalidad entre dos usuarios.Si ambos usuarios pertenecen al mismo grupo de personalidad, se aumenta el multiplicador. Si no, se reduce.
- const char * get_age_compatibility_level (int age_diff)

  Obtiene el nivel de compatibilidad basado en la diferencia de edad.
- double calculate_age_weight (int age1, int age2)

  Calcula el peso de la compatibilidad basado en la diferencia de edad, dependiendo de la diferencia de edad, se aplica un factor de penalización.
- void find_common_hobbies (const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int count1, const char hobbies2[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int count2)

  Encuentra los hobbies comunes entre dos usuarios, compara los hobbies de ambos usuarios e imprime los que son comunes.
- void quicksort (Match matches[ ], int low, int high)

  Ordena un arreglo de matches utilizando el algoritmo de quicksort, el arreglo se ordena de mayor a menor según la similitud entre los usuarios.
- int partition (Match matches[ ], int low, int high)

  Realiza la partición del arreglo para el algoritmo quicksort.

5.19.1 Detailed Description

Funciones para la comparación y recomendación de usuarios basadas en la personalidad, edad y hobbies.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamin Sanhueza y Johann Fink

Definition in file parameters.c.

5.19.2 Function Documentation

5.19.2.1 calculate_age_weight()

```
double calculate_age_weight (
            int age1,
            int age2)
```

Calcula el peso de la compatibilidad basado en la diferencia de edad, dependiendo de la diferencia de edad, se aplica un factor de penalización.

Calcula un factor de ponderación basado en la diferencia de edad entre dos usuarios. La diferencia de edad se convierte en un valor numérico para ajustar la similitud entre usuarios.

Parameters

age1	La edad del primer usuario.
age2	La edad del segundo usuario.

Returns

Un valor flotante que representa el peso de la compatibilidad basado en la diferencia de edad.

Calcula el peso de la compatibilidad basado en la diferencia de edad, dependiendo de la diferencia de edad, se aplica un factor de penalización.

```
int age_diff = abs(age1 - age2);
if (age_diff <= 5)
  return 1.0;
else if (age_diff <= 10)
  return 0.8;
else if (age_diff <= 15)
  return 0.6;
else if (age_diff <= 20)
  return 0.4;
else
  return 0.2;
```

Definition at line 149 of file parameters.c.

5.19.2.2 calculate_personality_multiplier()

```
double calculate_personality_multiplier (
            int group1,
            int group2)
```

Calcula el multiplicador de personalidad entre dos usuarios.Si ambos usuarios pertenecen al mismo grupo de personalidad, se aumenta el multiplicador. Si no, se reduce.

Calcula un multiplicador de personalidad entre dos usuarios. Compara las personalidades de dos usuarios y devuelve un multiplicador que indica cuán compatibles son sus personalidades.

Parameters

group1	El grupo de personalidad del primer usuario.
group2	El grupo de personalidad del segundo usuario.

Returns

Un valor flotante que representa el multiplicador de compatibilidad de personalidad.

Calcula el multiplicador de personalidad entre dos usuarios.Si ambos usuarios pertenecen al mismo grupo de personalidad, se aumenta el multiplicador. Si no, se reduce.

```
double personality_multiplier = 1.0;
if (group1 == group2)
   personality_multiplier = 1.2;
else if (group1 == 0 || group2 == 0)
  personality_multiplier = 1.0;
else
  personality_multiplier = 0.8;
return personality_multiplier;
```

Definition at line 82 of file parameters.c.

5.19.2.3 explain_personality_compatibility()

```
void explain_personality_compatibility (
            const User * user1,
            const User * user2)
```

Explica la compatibilidad de personalidad entre dos usuarios,Compara los grupos de personalidad de dos usuarios e imprime su nivel de compatibilidad.

Explica la compatibilidad de personalidad entre dos usuarios. Imprime una descripción detallada de cómo las personalidades de dos usuarios son compatibles.

Parameters

user1	El primer usuario.
user2	El segundo usuario.

Explica la compatibilidad de personalidad entre dos usuarios,Compara los grupos de personalidad de dos usuarios e imprime su nivel de compatibilidad.

```
const char *group_names[] = {"Sin grupo", "Analistas (Racionales)", "Diplomáticos (Idealistas)", "Centinelas
      (Conservadores)", "Exploradores (Artísticos)"};
int group1 = get_personality_group(user1->personality);
int group2 = get_personality_group(user2->personality);
if (group1 == 0 || group2 == 0)
  no tiene compatibilidad
if (group1 == group2)
  compatibilidad alta
else
  compatibilidad baja
```

Definition at line 43 of file parameters.c.

5.19.2.4 find_common_hobbies()

```
void find_common_hobbies (
            const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            int count1,
            const char hobbies2[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            int count2)
```

Encuentra los hobbies comunes entre dos usuarios, compara los hobbies de ambos usuarios e imprime los que son comunes.

Encuentra los hobbies comunes entre dos usuarios. Esta función compara los hobbies de dos usuarios y encuentra aquellos que coinciden.

Parameters

hobbies1	Lista de hobbies del primer usuario.
count1	El número de hobbies del primer usuario.
hobbies2	Lista de hobbies del segundo usuario.
count2	El número de hobbies del segundo usuario.

Encuentra los hobbies comunes entre dos usuarios, compara los hobbies de ambos usuarios e imprime los que son comunes

```
fprintf(stdout, "   - Hobbies en común: ");
int found_common = 0;
char seen[MAX_HOBBIES][MAX_HOBBIE_LENGTH];
int seen_count = 0;
for (int i = 0; i < count1; i++)
  Almacenar los hobbies del primer usuario
for (int j = 0; j < count2; j++)
  Verificar los hobbies del segundo usuario
if (!found_common)
   fprintf(stdout, "Ninguno");

fprintf(stdout, "\n");
```

Definition at line 187 of file parameters.c.

5.19.2.5 get_age_compatibility_level()

```
const char * get_age_compatibility_level (
             int age_diff)
```

Obtiene el nivel de compatibilidad basado en la diferencia de edad.

Obtiene el nivel de compatibilidad de edad entre dos usuarios. Esta función clasifica la compatibilidad de edad entre dos usuarios en categorías.

Parameters

age_diff	La diferencia de edad entre dos usuarios.

Returns

Una cadena de texto que indica el nivel de compatibilidad basado en la diferencia de edad.

Obtiene el nivel de compatibilidad basado en la diferencia de edad.

```
if (age_diff <= 5)
  return "Excelente";
else if (age_diff <= 10)
  return "Buena";
else if (age_diff <= 15)
  return "Moderada";
else if (age_diff <= 20)
  return "Baja";
else
 return "Muy baja";
```

Definition at line 114 of file parameters.c.

5.19.2.6 get_personality_group()

```
int get_personality_group (
             const char * personality)
```

Determina el grupo de personalidad de un usuario,compara las primeras tres letras de la personalidad del usuario y devuelve un grupo.

Obtiene el grupo de personalidad al que pertenece un usuario. La personalidad de un usuario se clasifica en un grupo de personalidades predefinido.

Parameters

personality	La personalidad del usuario.

Returns

El grupo de personalidad correspondiente (1: Analistas, 2: Diplomáticos, 3: Sentinelas, 4: Exploradores, 0 si no se encuentra).

Determina el grupo de personalidad de un usuario,compara las primeras tres letras de la personalidad del usuario y devuelve un grupo.

```
if (personality == NULL || strlen(personality) < 5) ->error
if (strncmp(personality, "INT", 3) == 0 || strncmp(personality, "ENT", 3) == 0)
  Compara las primeras 3 letras de la personalidad
```

Definition at line 14 of file parameters.c.

5.19.2.7 partition()

```
int partition (
            Match matches[],
            int low,
            int high)
```

Realiza la partición del arreglo para el algoritmo quicksort.

Particiona un arreglo de coincidencias para el algoritmo QuickSort. Se utiliza para reorganizar el arreglo y preparar el pivote para la ordenación.

Parameters

matches	El arreglo de matches a ordenar.
low	El índice más bajo del subarreglo.
high	El índice más alto del subarreglo.

Returns

El índice de partición que divide el arreglo en dos subarreglos.

```
double pivot = matches[high].similarity;
int i = low - 1;
for (int j = low; j < high; j++)
   Intercambiar elementos
Match temp = matches[i + 1];
matches[i + 1] = matches[high];
matches[high] = temp;
```

Definition at line 267 of file parameters.c.

5.19.2.8 quicksort()

```
void quicksort (
            Match matches[],
            int low,
            int high)
```

Ordena un arreglo de matches utilizando el algoritmo de quicksort, el arreglo se ordena de mayor a menor según la similitud entre los usuarios.

Ordena un arreglo de coincidencias de usuarios utilizando el algoritmo QuickSort. Este algoritmo organiza las coincidencias de acuerdo con su similitud, de mayor a menor.

Parameters

matches	El arreglo de matches a ordenar.
low	El índice más bajo del subarreglo.
high	El índice más alto del subarreglo.

Ordena un arreglo de matches utilizando el algoritmo de quicksort, el arreglo se ordena de mayor a menor según la similitud entre los usuarios.

```
if (low < high)
Ordena un arreglo de matches utilizando el algoritmo de quicksort, el arreglo se ordena de mayor a menor
      según la similitud entre los usuarios.
```

Definition at line 243 of file parameters.c.

5.20 parameters.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00014 int get_personality_group(const char *personality)
00015 {
00024     if (personality == NULL || strlen(personality) < 5)
00025         return 0;
00026
00027     if (strncmp(personality, "INT", 3) == 0 || strncmp(personality, "ENT", 3) == 0)
00028         return 1;
00029     else if (strncmp(personality, "INF", 3) == 0 || strncmp(personality, "ENF", 3) == 0)
00030         return 2;
00031     else if (strncmp(personality, "IST", 3) == 0 || strncmp(personality, "EST", 3) == 0)
00032         return 3;
00033     else if (strncmp(personality, "ISF", 3) == 0 || strncmp(personality, "ESF", 3) == 0)
00034         return 4;
00035
00036     return 0;
00037 }
00043 void explain_personality_compatibility(const User *user1, const User *user2)
00044 {
00059     const char *group_names[] = {"Sin grupo", "Analistas (Racionales)", "Diplomáticos (Idealistas)",
      "Centinelas (Conservadores)", "Exploradores (Artísticos)"};
00060
00061     int group1 = get_personality_group(user1->personality);
00062     int group2 = get_personality_group(user2->personality);
00063
00064     if (group1 == 0 || group2 == 0)
00065     {
00066         fprintf(stdout, "   - Compatibilidad no determinada \n");
00067         return;
00068     }
00069
00070     if (group1 == group2)
00071         fprintf(stdout, "   - Compatibilidad alta por mismo grupo: %s\n", group_names[group1]);
00072     else
00073         fprintf(stdout, "   - Compatibilidad BAJA por grupos diferentes: %s: %s - %s: %s\n",
      user1->username, group_names[group1], user2->username, group_names[group2]);
00074 }
00075
00082 double calculate_personality_multiplier(int group1, int group2)
00083 {
00097     double personality_multiplier = 1.0;
00098
00099     if (group1 == group2)
00100         personality_multiplier = 1.2;
00101     else if (group1 == 0 || group2 == 0)
00102         personality_multiplier = 1.0;
00103     else
00104         personality_multiplier = 0.8;
00105
00106     return personality_multiplier;
00107 }
00108
00114 const char *get_age_compatibility_level(int age_diff)
00115 {
00131     if (age_diff <= 5)
```

```
00132         return "Excelente";
00133     else if (age_diff <= 10)
00134         return "Buena";
00135     else if (age_diff <= 15)
00136         return "Moderada";
00137     else if (age_diff <= 20)
00138         return "Baja";
00139     else
00140         return "Muy baja";
00141 }
00142
00149 double calculate_age_weight(int age1, int age2)
00150 {
00167     int age_diff = abs(age1 - age2);
00168
00169     if (age_diff <= 5)
00170         return 1.0;
00171     else if (age_diff <= 10)
00172         return 0.8;
00173     else if (age_diff <= 15)
00174         return 0.6;
00175     else if (age_diff <= 20)
00176         return 0.4;
00177     else
00178         return 0.2;
00179 }
00187 void find_common_hobbies(const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int count1, const char
      hobbies2[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int count2)
00188 {
00206     fprintf(stdout, "   - Hobbies en común: ");
00207     int found_common = 0;
00208
00209     char seen[MAX_HOBBIES][MAX_HOBBIE_LENGTH];
00210     int seen_count = 0;
00211
00212     for (int i = 0; i < count1; i++)
00213         strcpy(seen[seen_count++], hobbies1[i]);
00214
00215     for (int j = 0; j < count2; j++)
00216     {
00217         for (int k = 0; k < seen_count; k++)
00218         {
00219             if (strcmp(hobbies2[j], seen[k]) == 0)
00220             {
00221                 if (found_common)
00222                     fprintf(stdout, ", ");
00223
00224                 fprintf(stdout, "%s", hobbies2[j]);
00225                 found_common = 1;
00226                 break;
00227             }
00228         }
00229     }
00230
00231     if (!found_common)
00232         fprintf(stdout, "Ninguno");
00233
00234     fprintf(stdout, "\n");
00235 }
00236
00243 void quicksort(Match matches[], int low, int high)
00244 {
00252     if (low < high)
00253     {
00254         int pivot_index = partition(matches, low, high);
00255
00256         quicksort(matches, low, pivot_index - 1);
00257         quicksort(matches, pivot_index + 1, high);
00258     }
00259 }
00267 int partition(Match matches[], int low, int high)
00268 {
00281     double pivot = matches[high].similarity;
00282     int i = low - 1;
00283
00284     for (int j = low; j < high; j++)
00285     {
00286         if (matches[j].similarity >= pivot)
00287         {
00288             i++;
00289             Match temp = matches[i];
00290             matches[i] = matches[j];
00291             matches[j] = temp;
00292         }
00293     }
00294
00295     Match temp = matches[i + 1];
```

```
00296     matches[i + 1] = matches[high];
00297     matches[high] = temp;
00298
00299     return i + 1;
00300 }
```

5.21 src/post.c File Reference

Implementación de funciones para la gestión y visualización de publicaciones en la red social.

```
#include "main.h"
```

Functions

- void init_post_list (Post_List *post_list)

  Inicializa la lista de publicaciones, configura el encabezado de la lista de publicaciones a NULL y establece el contador de publicaciones en 0.
- Post * create_post (int user_Id, const char *username, const char *content)

  Crea una nueva publicación,esta función genera una nueva publicación con un ID único.
- void publish_post (Post_List *post_list, const User *user, const char *content)

  Publica una nueva entrada en el muro global-Crea una publicación usando la información de un usuario y agrega la publicación al inicio de la lista global de publicaciones.
- void display_all_posts (const Post_List *post_list)

  Muestra todas las publicaciones en la red social. Imprime en consola la lista completa de publicaciones, mostrando la ID, el usuario, el contenido y la fecha.
- void free_all_posts (Post_List *post_list)

  Libera la memoria ocupada por todas las publicaciones. Recorre la lista de publicaciones y libera la memoria de cada una de ellas.
- void load_post_templates (char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH], int *post_count)

  Carga plantillas de publicaciones desde un archivo. Lee un archivo de plantillas de publicaciones y almacena las plantillas en un arreglo.
- void generate_random_posts (User users[MAX_USERS], int num_users, Post_List *post_list)

  Genera publicaciones aleatorias usando usuarios y plantillas. Esta función selecciona aleatoriamente usuarios y plantillas de publicaciones, creando publicaciones y agregándolas al muro global hasta alcanzar el número máximo de publicaciones.
- time_t generate_random_timestamp ()

  Genera un timestamp aleatorio. Calcula un timestamp aleatorio basado en el tiempo actual, con una desviación de hasta 7 días.

5.21.1 Detailed Description

Implementación de funciones para la gestión y visualización de publicaciones en la red social.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamin Sanhueza y Johann Fink

Definition in file post.c.

5.21.2 Function Documentation

5.21.2.1 create_post()

```
Post * create_post (
            int user_Id,
            const char * username,
            const char * content)
```

Crea una nueva publicación,esta función genera una nueva publicación con un ID único.

Crea una nueva publicación. Esta función crea una nueva publicación, asignando un ID único, y copiando el nombre de usuario y contenido proporcionado. La marca de tiempo es generada al momento de la creación.

Parameters

user_Id	ID del usuario que realiza la publicación.
username	Nombre de usuario del autor de la publicación.
content	Contenido textual de la publicación.

Returns

Un puntero a la nueva publicación creada.

Crea una nueva publicación,esta función genera una nueva publicación con un ID único.

```
if (strlen(content) >= MAX_POST_LENGTH) ->error
Post *newPost = malloc(sizeof(Post));
if (!newPost) ->error
static int post_Id_Counter = 1;
newPost->post_Id = post_Id_Counter++;
newPost->user_Id = user_Id;
strncpy(newPost->username, username, MAX_NAME_LENGTH - 1);
newPost->username[MAX_NAME_LENGTH - 1] = '\0';
strncpy(newPost->content, content, MAX_POST_LENGTH - 1);
newPost->content[MAX_POST_LENGTH - 1] = '\0';
newPost->timestamp = generate_random_timestamp();
newPost->next = NULL;
```

Definition at line 33 of file post.c.

5.21.2.2 display_all_posts()

```
void display_all_posts (
            const Post_List * post_list)
```

Muestra todas las publicaciones en la red social. Imprime en consola la lista completa de publicaciones, mostrando la ID, el usuario, el contenido y la fecha.

Muestra todas las publicaciones en la lista. Esta función imprime el contenido de todas las publicaciones almacenadas en la lista.

Parameters

post_list	Puntero a la lista de publicaciones que se mostrarán.

Muestra todas las publicaciones en la red social. Imprime en consola la lista completa de publicaciones, mostrando la ID, el usuario, el contenido y la fecha.

```
fprintf(stdout, RED "\nPublicaciones:\n\n" RESET);
Post *current = post_list->head;
while (current)
   Imprime en consola la lista completa de publicaciones, mostrando la ID, el usuario, el contenido y la
      fecha.
```

Definition at line 110 of file post.c.

5.21.2.3 free_all_posts()

```
void free_all_posts (
            Post_List * post_list)
```

Libera la memoria ocupada por todas las publicaciones. Recorre la lista de publicaciones y libera la memoria de cada una de ellas.

Libera la memoria ocupada por todas las publicaciones de la lista. Esta función recorre la lista de publicaciones y libera la memoria de cada publicación.

Parameters

post_list	Puntero a la lista de publicaciones que se deben liberar.

Libera la memoria ocupada por todas las publicaciones. Recorre la lista de publicaciones y libera la memoria de cada una de ellas.

```
Post *current = post_list->head;
while (current)
  Libera la memoria ocupada
post_list->head = NULL;
post_list->postCount = 0;
```

Definition at line 143 of file post.c.

5.21.2.4 generate_random_posts()

```
void generate_random_posts (
            User users[MAX_USERS],
            int num_users,
            Post_List * post_list)
```

Genera publicaciones aleatorias usando usuarios y plantillas. Esta función selecciona aleatoriamente usuarios y plantillas de publicaciones, creando publicaciones y agregándolas al muro global hasta alcanzar el número máximo de publicaciones.

Genera publicaciones aleatorias para un conjunto de usuarios. Esta función genera un número de publicaciones aleatorias para los usuarios en el arreglo `users`.

Parameters

users	Arreglo de usuarios disponibles para generar publicaciones.
num_users	Número de usuarios disponibles.
post_list	Puntero a la lista de publicaciones donde se agregarán las nuevas publicaciones generadas.

Genera publicaciones aleatorias usando usuarios y plantillas.

```
char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH];
int post_template_count = 0;
load_post_templates(post_templates, &post_template_count);
if (post_template_count == 0) ->error
int used_users[MAX_USERS] = {0};
int unique_posts_created = 0;
while (unique_posts_created < MAX_POSTS && unique_posts_created < num_users)
  Generar publicaciones aleatorias hasta alcanzar el límite o usar todos los usuarios
```

Definition at line 220 of file post.c.

5.21.2.5 generate_random_timestamp()

```
time_t generate_random_timestamp ()
```

Genera un timestamp aleatorio. Calcula un timestamp aleatorio basado en el tiempo actual, con una desviación de hasta 7 días.

Genera una marca de tiempo aleatoria. Esta función genera una marca de tiempo aleatoria para simular el momento de publicación de una publicación.

Returns

El timestamp generado aleatoriamente.

Genera un timestamp aleatorio.

```
time_t current_time = time(NULL);
int random_hours = rand() % (24 * 7);
int random_minutes = rand() % 60;
int random_seconds = rand() % 60;
time_t random_time = current_time - (random_hours * 3600 + random_minutes * 60 + random_seconds);
return random_time;
```

Definition at line 272 of file post.c.

5.21.2.6 init_post_list()

```
void init_post_list (
            Post_List * post_list)
```

Inicializa la lista de publicaciones, configura el encabezado de la lista de publicaciones a NULL y establece el contador de publicaciones en 0.

Inicializa una lista de publicaciones. Esta función inicializa la lista de publicaciones configurando el puntero `head` a NULL y el contador postCount` a 0.

Parameters

post_list	Puntero a la lista de publicaciones a inicializar.

Inicializa la lista de publicaciones, configura el encabezado de la lista de publicaciones a NULL y establece el contador de publicaciones en 0.

```
post_list->head = NULL;
post_list->postCount = 0;
```

Definition at line 13 of file post.c.

5.21.2.7 load_post_templates()

```
void load_post_templates (
            char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH],
            int * post_count)
```

Carga plantillas de publicaciones desde un archivo. Lee un archivo de plantillas de publicaciones y almacena las plantillas en un arreglo.

Carga plantillas de publicaciones desde un archivo. Esta función carga las plantillas de publicaciones desde un archivo de texto, donde cada línea corresponde a una plantilla.

Parameters

post_templates	Arreglo donde se almacenarán las plantillas de publicaciones.
post_count	Puntero a la variable que contará las plantillas cargadas.

Carga plantillas de publicaciones desde un archivo.

```
FILE *file = fopen("./input/post_templates.txt", "r");
if (!file) ->error
*post_count = 0;
char line[MAX_POST_LENGTH];
while (*post_count < MAX_FILE_LINES && fgets(line, sizeof(line), file))
  Carga plantillas de publicaciones desde un archivo.
```

Definition at line 174 of file post.c.

5.21.2.8 publish_post()

```
void publish_post (
            Post_List * post_list,
            const User * user,
            const char * content)
```

Publica una nueva entrada en el muro global-Crea una publicación usando la información de un usuario y agrega la publicación al inicio de la lista global de publicaciones.

Publica un nuevo post en la lista. Esta función agrega una nueva publicación al final de la lista de publicaciones.

Parameters

post_list	Puntero a la lista de publicaciones global.
user	Puntero a la estructura del usuario que realiza la publicación.
content	Contenido textual de la publicación.

Publica una nueva entrada en el muro global-Crea una publicación usando la información de un usuario y agrega la publicación al inicio de la lista global de publicaciones.

```
Post *newPost = create_post(user->id, user->username, content);
if (newPost)
  Agregar al inicio de la lista global
```

Definition at line 86 of file post.c.

5.22 post.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00013 void init_post_list(Post_List *post_list)
00014 {
00022     post_list->head = NULL;
00023     post_list->postCount = 0;
00024 }
00025
00033 Post *create_post(int user_Id, const char *username, const char *content)
00034 {
00052     if (strlen(content) >= MAX_POST_LENGTH)
00053     {
00054         fprintf(stderr, "El contenido de publicación demasiado largo. Saliendo...\n");
00055         exit(EXIT_FAILURE);
```

```
00056     }
00057
00058     Post *newPost = malloc(sizeof(Post));
00059     if (!newPost)
00060     {
00061         fprintf(stderr, "No se pudo asignar memoria para la publicación. Saliendo...\n");
00062         exit(EXIT_FAILURE);
00063     }
00064
00065     static int post_Id_Counter = 1;
00066     newPost->post_Id = post_Id_Counter++;
00067
00068     newPost->user_Id = user_Id;
00069     strncpy(newPost->username, username, MAX_NAME_LENGTH - 1);
00070     newPost->username[MAX_NAME_LENGTH - 1] = '\0';
00071     strncpy(newPost->content, content, MAX_POST_LENGTH - 1);
00072     newPost->content[MAX_POST_LENGTH - 1] = '\0';
00073
00074     newPost->timestamp = generate_random_timestamp();
00075     newPost->next = NULL;
00076
00077     return newPost;
00078 }
00079
00086 void publish_post(Post_List *post_list, const User *user, const char *content)
00087 {
00097     Post *newPost = create_post(user->id, user->username, content);
00098     if (newPost)
00099     {
00100         newPost->next = post_list->head;
00101         post_list->head = newPost;
00102         post_list->postCount++;
00103     }
00104 }
00105
00110 void display_all_posts(const Post_List *post_list)
00111 {
00121     fprintf(stdout, RED "\nPublicaciones:\n\n" RESET);
00122
00123     Post *current = post_list->head;
00124     while (current)
00125     {
00126         char timeStr[64];
00127         struct tm *tm_info = localtime(&current->timestamp);
00128         strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", tm_info);
00129
00130         fprintf(stdout, CYAN "Publicación %d" RESET "\n", current->post_Id);
00131         fprintf(stdout, GREEN "Usuario: %s (ID: %d)\n" RESET, current->username, current->user_Id);
00132         fprintf(stdout, YELLOW "Contenido: %s\n" RESET, current->content);
00133         fprintf(stdout, "Fecha: %s\n", timeStr);
00134         fprintf(stdout, "\n");
00135         current = current->next;
00136     }
00137 }
00138
00143 void free_all_posts(Post_List *post_list)
00144 {
00155     Post *current = post_list->head;
00156
00157     while (current)
00158     {
00159         Post *temp = current;
00160         current = current->next;
00161         free(temp);
00162     }
00163
00164     post_list->head = NULL;
00165     post_list->postCount = 0;
00166 }
00167
00174 void load_post_templates(char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH], int *post_count)
00175 {
00187     FILE *file = fopen("./input/post_templates.txt", "r");
00188     if (!file)
00189     {
00190         fprintf(stderr, "Error al abrir el archivo post_templates.txt, saliendo...\n");
00191         exit(EXIT_FAILURE);
00192     }
00193
00194     *post_count = 0;
00195     char line[MAX_POST_LENGTH];
00196
00197     while (*post_count < MAX_FILE_LINES && fgets(line, sizeof(line), file))
00198     {
00199         size_t len = strcspn(line, "\n");
00200         line[len] = '\0';
00201
```

```
00202         if (len > MAX_POST_LENGTH)
00203             len = MAX_POST_LENGTH - 1;
00204
00205         memcpy(post_templates[*post_count], line, len);
00206         post_templates[*post_count][len] = '\0';
00207         (*post_count)++;
00208     }
00209
00210     fclose(file);
00211 }
00212
00220 void generate_random_posts(User users[MAX_USERS], int num_users, Post_List *post_list)
00221 {
00235     char post_templates[MAX_FILE_LINES][MAX_POST_LENGTH];
00236     int post_template_count = 0;
00237
00238     load_post_templates(post_templates, &post_template_count);
00239
00240     if (post_template_count == 0)
00241     {
00242         fprintf(stderr, "No se encontraron plantillas de publicaciones.\n");
00243         exit(EXIT_FAILURE);
00244     }
00245
00246     int used_users[MAX_USERS] = {0};
00247     int unique_posts_created = 0;
00248
00249     while (unique_posts_created < MAX_POSTS && unique_posts_created < num_users)
00250     {
00251         int random_user_index;
00252         do
00253         {
00254             random_user_index = rand() % num_users;
00255         } while (used_users[random_user_index]);
00256
00257         used_users[random_user_index] = 1;
00258
00259         int template_index = rand() % post_template_count;
00260
00261         publish_post(post_list, &users[random_user_index], post_templates[template_index]);
00262
00263         unique_posts_created++;
00264     }
00265 }
00266
00272 time_t generate_random_timestamp()
00273 {
00285     time_t current_time = time(NULL);
00286     int random_hours = rand() % (24 * 7);
00287     int random_minutes = rand() % 60;
00288     int random_seconds = rand() % 60;
00289     time_t random_time = current_time - (random_hours * 3600 + random_minutes * 60 + random_seconds);
00290
00291     return random_time;
00292 }
```

5.23 src/search.c File Reference

Funciones para la busqueda del usuario con mas amigos.

```
#include "main.h"
```

Functions

- int find_user_with_most_friends (Graph *graph)

  Encuentra al usuario con más amigos en el grafo. Recorre el grafo y cuenta las conexiones (amigos) de cada usuario, devolviendo el índice del usuario con más amigos.

- void print_friends_of_user (Graph *graph, int userIndex)

  Imprime los amigos de un usuario dado en el grafo. Muestra en pantalla los amigos (conexiones) del usuario especificado por su índice en el grafo.

5.23.1 Detailed Description

Funciones para la busqueda del usuario con mas amigos.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamin Sanhueza y Johann Fink

Definition in file search.c.

5.23.2 Function Documentation

5.23.2.1 find_user_with_most_friends()

```
int find_user_with_most_friends (
            Graph * graph)
```

Encuentra al usuario con más amigos en el grafo. Recorre el grafo y cuenta las conexiones (amigos) de cada usuario, devolviendo el índice del usuario con más amigos.

Encuentra el usuario con más amigos en el grafo.

Parameters

graph	Puntero al grafo de usuarios donde se buscan los amigos.

Returns

El índice del usuario con más amigos. Si no se encuentra ningún usuario, devuelve -1.

Encuentra al usuario con más amigos en el grafo.

```
int maxFriends = -1;
int userIndex = -1;
int i;
if (!graph || graph->numUsers == 0) ->error
for (i = 0; i < graph->numUsers; i++)
     int friendCount = 0;
     Node *current = graph->adjacencyList[i];
     while (current)
         Contar las conexiones del usuario actual
     if (friendCount > maxFriends)
         Contar las conexiones del usuario actual
     if (friendCount > maxFriends)
         Actualizar si el usuario actual tiene más amigos
if (userIndex != -1)->muestra usuario con mas amigos
else -> no se encontraron usuarios con amigos
```

Definition at line 15 of file search.c.

5.23.2.2 print_friends_of_user()

```
void print_friends_of_user (
            Graph * graph,
            int userIndex)
```

Imprime los amigos de un usuario dado en el grafo. Muestra en pantalla los amigos (conexiones) del usuario especificado por su índice en el grafo.

Encuentra el usuario con menos amigos en el grafo.

Parameters

graph	Puntero al grafo que contiene la información de los usuarios y sus amigos.
userIndex	Índice del usuario en el grafo del cual se desean imprimir los amigos.

Imprime los amigos de un usuario dado en el grafo.

```
if (!graph || userIndex < 0 || userIndex >= graph->numUsers) ->error
Node *current = graph->adjacencyList[userIndex];
fprintf(stdout, CYAN "Amigos de %s:\n" RESET, graph->user_names[userIndex]);
while (current)
  imprime los amigos del usuario con mas amigos
```

Definition at line 79 of file search.c.

5.24 search.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00015 int find_user_with_most_friends(Graph *graph)
00016 {
00037     int maxFriends = -1;
00038     int userIndex = -1;
00039     int i;
00040
00041     if (!graph || graph->numUsers == 0)
00042     {
00043         fprintf(stderr, RED "El grafo está vacío o no inicializado.\n" RESET);
00044         exit(EXIT_FAILURE);
00045     }
00046
00047     for (i = 0; i < graph->numUsers; i++)
00048     {
00049         int friendCount = 0;
00050         Node *current = graph->adjacencyList[i];
00051
00052         while (current)
00053         {
00054             friendCount++;
00055             current = current->next;
00056         }
00057
00058         if (friendCount > maxFriends)
00059         {
00060             maxFriends = friendCount;
00061             userIndex = i;
00062         }
00063     }
00064
00065     if (userIndex != -1)
00066         fprintf(stdout, GREEN "\nEl usuario con más amigos es: %s con %d amigos.\n\n" RESET,
      graph->user_names[userIndex], maxFriends);
00067     else
00068         fprintf(stdout, YELLOW "No se encontraron usuarios con amigos.\n" RESET);
00069
00070     return userIndex;
00071 }
00072
00079 void print_friends_of_user(Graph *graph, int userIndex)
00080 {
00091     if (!graph || userIndex < 0 || userIndex >= graph->numUsers)
00092     {
00093         fprintf(stderr, RED "Índice de usuario inválido o grafo no inicializado.\n" RESET);
00094         return;
00095     }
00096
00097     Node *current = graph->adjacencyList[userIndex];
00098     fprintf(stdout, CYAN "Amigos de %s:\n" RESET, graph->user_names[userIndex]);
00099
00100     while (current)
00101     {
00102         fprintf(stdout, "- %s\n", graph->user_names[current->id]);
00103         current = current->next;
00104     }
00105 }
```

5.25 src/similarity.c File Reference

Contiene funciones para calcular similitudes entre usuarios basadas en sus hobbies, edad y personalidad, así como para crear conexiones entre usuarios mediante un grafo y recomendar usuarios similares.

```
#include "main.h"
```

Functions

- double calculate_jaccard_similarity (const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int count1, const char hobbies2[MAX_HOBBIE_LENGTH][MAX_HOBBIE_LENGTH], int count2, int age1, int age2, const char ∗personality1, const char ∗personality2)

  Calcula el índice de similitud de Jaccard entre dos conjuntos de hobbies. Compara dos usuarios basándose en sus hobbies, edad y personalidad, ajustando el puntaje de similitud según estos factores.
- void recommend_users (const User users[MAX_USERS], int num_users)

  Recomienda usuarios basándose en la similitud de hobbies, edad y personalidad. Compara un usuario con todos los demás y muestra los usuarios más similares.
- void create_connections (const User users[MAX_USERS], int num_users, Graph ∗graph, double threshold)

  Crea conexiones entre usuarios que tienen un índice de similitud de Jaccard por encima de un umbral. Recorre todos los usuarios y crea conexiones entre ellos si su similitud de Jaccard es mayor o igual al umbral.

5.25.1 Detailed Description

Contiene funciones para calcular similitudes entre usuarios basadas en sus hobbies, edad y personalidad, así como para crear conexiones entre usuarios mediante un grafo y recomendar usuarios similares.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamin Sanhueza y Johann Fink

Definition in file similarity.c.

5.25.2 Function Documentation

5.25.2.1 calculate_jaccard_similarity()

```
double calculate_jaccard_similarity (
            const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            int count1,
            const char hobbies2[MAX_HOBBIE_LENGTH][MAX_HOBBIE_LENGTH],
            int count2,
            int age1,
            int age2,
            const char * personality1,
            const char * personality2)
```

Calcula el índice de similitud de Jaccard entre dos conjuntos de hobbies. Compara dos usuarios basándose en sus hobbies, edad y personalidad, ajustando el puntaje de similitud según estos factores.

Calcula la similitud de Jaccard entre dos conjuntos de hobbies. Esta función calcula la similitud de Jaccard entre los hobbies de dos usuarios, tomando en cuenta el número de hobbies comunes y el número total de hobbies.

Parameters

hobbies1	Conjunto de hobbies del primer usuario.
count1	Número de hobbies del primer usuario.
hobbies2	Conjunto de hobbies del segundo usuario.
count2	Número de hobbies del segundo usuario.
age1	Edad del primer usuario.
age2	Edad del segundo usuario.
personality1	Personalidad del primer usuario.
personality2	Personalidad del segundo usuario.

Returns

Un valor entre 0 y 1 que representa la similitud entre los dos usuarios.

Calcula el índice de similitud de Jaccard entre dos conjuntos de hobbies.

```
int intersection = 0, union_count = 0;
char seen[MAX_HOBBIES][MAX_HOBBIE_LENGTH] = {0};
int seen_count = 0;
for (int i = 0; i < count1; i++)
  for (int j = 0; j < count2; j++)
      contar interseccion
for (int i = 0; i < count1; i++)
for (int j = 0; j < count2; j++)
  contar union
union_count = seen_count;
double jaccard = union_count > 0 ? (double)intersection / union_count : 0.0;
double age_weight = calculate_age_weight(age1, age2);
int group1 = get_personality_group(personality1);
int group2 = get_personality_group(personality2);
double personality_multiplier = calculate_personality_multiplier(group1, group2);
return jaccard * age_weight * personality_multiplier; -> Retorna el puntaje ajustado por edad y el
     multiplicador de personalidad
```

Definition at line 22 of file similarity.c.

5.25.2.2 create_connections()

```
void create_connections (
            const User users[MAX_USERS],
            int num_users,
            Graph * graph,
            double threshold)
```

Crea conexiones entre usuarios que tienen un índice de similitud de Jaccard por encima de un umbral. Recorre todos los usuarios y crea conexiones entre ellos si su similitud de Jaccard es mayor o igual al umbral.

Genera conexiones aleatorias entre usuarios en el grafo.

Parameters

users	Arreglo de usuarios a evaluar.
num_users	Número total de usuarios.
graph	Grafo donde se almacenarán las conexiones.
threshold	Valor mínimo de similitud de Jaccard para crear una conexión.

Crea conexiones entre usuarios que tienen un índice de similitud de Jaccard por encima de un umbral.

```
int connections_found = 0;
for (int i = 0; i < num_users; i++)
  for (int j = i + 1; j < num_users; j++)
       int count1 = 0, count2 = 0;
       double similarity = calculate_jaccard_similarity(users[i].hobbies, count1, users[j].hobbies, count2,
      users[i].age, users[j].age, users[i].personality, users[j].personality);
       if (similarity >= threshold)
           connections_found = 1;
           fprintf(stdout, CYAN "\nConectando a los usuarios %s y %s (Índice de Jaccard: %.2f)\n" RESET,
      users[i].username, users[j].username, similarity);
           add_connection(graph, i, j);
if (!connections_found) ->error
```

Definition at line 178 of file similarity.c.

5.25.2.3 recommend_users()

```
void recommend_users (
             const User users[MAX_USERS],
             int num_users)
```

Recomienda usuarios basándose en la similitud de hobbies, edad y personalidad. Compara un usuario con todos los demás y muestra los usuarios más similares.

Recomienda usuarios basándose en la similitud. Compara todos los usuarios entre sí y recomienda aquellos con mayor compatibilidad basada en la similitud de hobbies, edad y personalidad.

Parameters

users	Arreglo de usuarios a evaluar.
num_users	Número total de usuarios.

Recomienda usuarios basándose en la similitud de hobbies, edad y personalidad.

```
for (int i = 0; i < num_users; i++)
 fprintf(stdout, CYAN "\nUsuario %d (%s, %d años):" RESET, users[i].id, users[i].username, users[i].age);
 Match matches[MAX_USERS];
 int match_count = 0;
 int count1 = 0, count2 = 0;
 for (int k = 0; k < MAX_HOBBIES && strlen(users[i].hobbies[k]) > 0; k++)
      Calcular el número de hobbies para el usuario i.
 for (int j = 0; j < num_users; j++)
      Encontrar todas las coincidencias.
 if (match_count > 1) -> Ordenar las coincidencias por similitud (método quicksort).
 int show_matches = match_count > 3 ? 3 : match_count;
 if (show_matches > 0) -> Mostrar las mejores coincidencias (hasta 3)
```

Definition at line 94 of file similarity.c.

5.26 similarity.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00022 double calculate_jaccard_similarity(const char hobbies1[MAX_HOBBIES][MAX_HOBBIE_LENGTH], int count1,
      const char hobbies2[MAX_HOBBIE_LENGTH][MAX_HOBBIE_LENGTH], int count2, int age1, int age2, const char
      *personality1, const char *personality2)
00023 {
00045     int intersection = 0, union_count = 0;
```

```
00046
00047     char seen[MAX_HOBBIES][MAX_HOBBIE_LENGTH] = {0};
00048     int seen_count = 0;
00049
00050     for (int i = 0; i < count1; i++)
00051         for (int j = 0; j < count2; j++)
00052             if (strcmp(hobbies1[i], hobbies2[j]) == 0)
00053             {
00054                 intersection++;
00055                 break;
00056             }
00057
00058     for (int i = 0; i < count1; i++)
00059         strcpy(seen[seen_count++], hobbies1[i]);
00060
00061     for (int j = 0; j < count2; j++)
00062     {
00063         int found = 0;
00064         for (int k = 0; k < seen_count; k++)
00065             if (strcmp(hobbies2[j], seen[k]) == 0)
00066             {
00067                 found = 1;
00068                 break;
00069             }
00070
00071         if (!found)
00072             strcpy(seen[seen_count++], hobbies2[j]);
00073     }
00074
00075     union_count = seen_count;
00076
00077     double jaccard = union_count > 0 ? (double)intersection / union_count : 0.0;
00078     double age_weight = calculate_age_weight(age1, age2);
00079
00080     int group1 = get_personality_group(personality1);
00081     int group2 = get_personality_group(personality2);
00082
00083     double personality_multiplier = calculate_personality_multiplier(group1, group2);
00084
00085     return jaccard * age_weight * personality_multiplier;
00086 }
00087
00094 void recommend_users(const User users[MAX_USERS], int num_users)
00095 {
00113     for (int i = 0; i < num_users; i++)
00114     {
00115         fprintf(stdout, CYAN "\nUsuario %d (%s, %d años):" RESET, users[i].id, users[i].username,
      users[i].age);
00116
00117         Match matches[MAX_USERS];
00118         int match_count = 0;
00119         int count1 = 0, count2 = 0;
00120
00121         for (int k = 0; k < MAX_HOBBIES && strlen(users[i].hobbies[k]) > 0; k++)
00122             count1++;
00123
00124         for (int j = 0; j < num_users; j++)
00125         {
00126             if (i == j)
00127                 continue;
00128
00129             count2 = 0;
00130
00131             for (int k = 0; k < MAX_HOBBIES && strlen(users[j].hobbies[k]) > 0; k++)
00132                 count2++;
00133
00134             double similarity = calculate_jaccard_similarity(users[i].hobbies, count1,
      users[j].hobbies, count2, users[i].age, users[j].age, users[i].personality, users[j].personality);
00135
00136             if (similarity > 0)
00137             {
00138                 matches[match_count].user_index = j;
00139                 matches[match_count].similarity = similarity;
00140                 matches[match_count].age_diff = abs(users[i].age - users[j].age);
00141                 match_count++;
00142             }
00143         }
00144
00145         if (match_count > 1)
00146             quicksort(matches, 0, match_count - 1);
00147
00148         int show_matches = match_count > 3 ? 3 : match_count;
00149
00150         if (show_matches > 0)
00151         {
00152             fprintf(stdout, "\nMejores coincidencias:\n");
00153
```

```
00154             for (int m = 0; m < show_matches; m++)
00155             {
00156                 int j = matches[m].user_index;
00157 
00158                 fprintf(stdout, GREEN "\n%d. Nombre: %s (%d años)\n" RESET, m + 1, users[j].username,
      users[j].age);
00159                 fprintf(stdout, YELLOW "   - Similitud total: %.2f\n" RESET, matches[m].similarity);
00160                 fprintf(stdout, "   - Compatibilidad por edad: %s (diferencia %d años)\n",
      get_age_compatibility_level(matches[m].age_diff), matches[m].age_diff);
00161                 explain_personality_compatibility(&users[i], &users[j]);
00162                 find_common_hobbies(users[i].hobbies, count1, users[j].hobbies, count2);
00163             }
00164         }
00165         else
00166             fprintf(stdout, " No hay usuarios recomendados.\n");
00167     }
00168 }
00169 
00178 void create_connections(const User users[MAX_USERS], int num_users, Graph *graph, double threshold)
00179 {
00195     int connections_found = 0;
00196 
00197     for (int i = 0; i < num_users; i++)
00198         for (int j = i + 1; j < num_users; j++)
00199         {
00200             int count1 = 0, count2 = 0;
00201 
00202             for (int k = 0; k < MAX_HOBBIES && strlen(users[i].hobbies[k]) > 0; k++)
00203                 count1++;
00204 
00205             for (int k = 0; k < MAX_HOBBIES && strlen(users[j].hobbies[k]) > 0; k++)
00206                 count2++;
00207 
00208             double similarity = calculate_jaccard_similarity(users[i].hobbies, count1,
      users[j].hobbies, count2, users[i].age, users[j].age, users[i].personality, users[j].personality);
00209 
00210             if (similarity >= threshold)
00211             {
00212                 connections_found = 1;
00213                 fprintf(stdout, CYAN "\nConectando a los usuarios %s y %s (Índice de Jaccard: %.2f)\n"
      RESET, users[i].username, users[j].username, similarity);
00214                 add_connection(graph, i, j);
00215             }
00216         }
00217 
00218     if (!connections_found)
00219         fprintf(stdout, CYAN "\nNingún usuario con Índice de Jaccard por encima de %.2f\n" RESET,
      threshold);
00220 }
```

5.27 src/user.c File Reference

Implementación de funciones para generar y gestionar usuarios. Incluye la creación de usuarios aleatorios, carga de archivos con información de usuarios, y la impresión de los datos de los usuarios.

```
#include "main.h"
```

Functions

- void load_file (const char *filename, char file_array[MAX_FILE_LINES][MAX_NAME_LENGTH], int *count)

  Carga un archivo de texto en un arreglo. Esta función lee un archivo de texto y almacena cada línea en un arreglo bidimensional, hasta un máximo de líneas especificado por MAX_FILE_LINES.

- void generate_random_users (User *user, int id, char male_usernames[MAX_FILE_LINES][MAX_↩ NAME_LENGTH], int male_count, char female_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH], int female_count, char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int hobby_count, char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH], int personality_count)

  Genera un usuario aleatorio. Esta función crea un usuario con información aleatoria como el género, nombre, edad, personalidad y hobbies, basándose en listas de datos predefinidas.

- void generate_random_hobbies (char hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH], char hobbies_↩ list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int hobby_count)

  Genera hobbies aleatorios para un usuario. Esta función selecciona aleatoriamente un número de hobbies de una lista predefinida, y los asigna al arreglo de hobbies del usuario.
- void generate_random_personality (char ∗personality, char personalities_list[MAX_FILE_LINES][MAX_↩ PERS_LENGTH], int personality_count)

  Genera una personalidad aleatoria para un usuario. Esta función selecciona aleatoriamente una personalidad de una lista predefinida y la asigna al campo de personalidad del usuario.
- void print_users (const User ∗user)

  Imprime la información de un usuario. Esta función imprime en la salida estándar (consola) los detalles de un usuario, incluyendo su ID, nombre, género, edad, personalidad y hobbies.

5.27.1 Detailed Description

Implementación de funciones para generar y gestionar usuarios. Incluye la creación de usuarios aleatorios, carga de archivos con información de usuarios, y la impresión de los datos de los usuarios.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamin Sanhueza y Johann Fink

Definition in file user.c.

5.27.2 Function Documentation

5.27.2.1 generate_random_hobbies()

```
void generate_random_hobbies (
            char hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH],
            char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH],
            int hobby_count)
```

Genera hobbies aleatorios para un usuario. Esta función selecciona aleatoriamente un número de hobbies de una lista predefinida, y los asigna al arreglo de hobbies del usuario.

Genera hobbies aleatorios para un usuario. Selecciona aleatoriamente un número de hobbies de una lista predefinida y los asigna al usuario.

Parameters

hobbies	Arreglo donde se almacenarán los hobbies generados.
hobbies_list	Lista de hobbies disponibles para selección.
hobby_count	Número total de hobbies disponibles en la lista.

Genera hobbies aleatorios para un usuario.

```
*if (hobby_count <= 0) ->Error
for (int i = 0; i < MAX_HOBBIES; i++)
int *hobbie_selected = calloc(hobby_count, sizeof(int));
if (!hobbie_selected) ->return
int num_hobbies = (random() % MAX_HOBBIES) + 1;
int added_hobbies = 0;
for (int i = 0; i < num_hobbies && added_hobbies < MAX_HOBBIES; i++)
 Selecciona aleatoriamente hobbies únicos de una lista disponible, los asigna al usuario y asegura que no se
      exceda el número máximo permitido.
```

Definition at line 130 of file user.c.

5.27.2.2 generate_random_personality()

```
void generate_random_personality (
            char * personality,
            char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH],
            int personality_count)
```

Genera una personalidad aleatoria para un usuario. Esta función selecciona aleatoriamente una personalidad de una lista predefinida y la asigna al campo de personalidad del usuario.

Genera una personalidad aleatoria para un usuario. Selecciona aleatoriamente una personalidad de una lista predefinida y la asigna al usuario.

Parameters

personality	Puntero a una cadena donde se almacenará la personalidad generada.
personalities_list	Lista de personalidades disponibles para selección.
personality_count	Número total de personalidades disponibles en la lista.

Genera una personalidad aleatoria para un usuario.

```
  if (personality_count <= 0) ->error
  personality[0] = '\0';
*strcpy(personality, personalities_list[random() % personality_count]);
```

Definition at line 186 of file user.c.

5.27.2.3 generate_random_users()

```
void generate_random_users (
            User * user,
            int id,
            char male_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH],
            int male_count,
            char female_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH],
            int female_count,
            char hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH],
            int hobby_count,
            char personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH],
            int personality_count)
```

Genera un usuario aleatorio. Esta función crea un usuario con información aleatoria como el género, nombre, edad, personalidad y hobbies, basándose en listas de datos predefinidas.

Genera un usuario aleatorio. Crea un usuario con información aleatoria, como nombre, edad, género, hobbies y personalidad, seleccionados de listas predefinidas.

Parameters

user	Puntero al usuario que se generará.
id	ID único que se asignará al usuario.
male_usernames	Lista de nombres de usuario masculinos disponibles.
male_count	Número de nombres masculinos en la lista.
female_usernames	Lista de nombres de usuario femeninos disponibles.
female_count	Número de nombres femeninos en la lista.
hobbies_list	Lista de hobbies disponibles.

hobby_count	Número de hobbies en la lista.
personalities_list	Lista de personalidades disponibles.
personality_count	Número de personalidades en la lista.

Genera un usuario aleatorio.

```
if (!user) ->Error
user->id = id;
user->age = random() % MAX_AGE + MIN_AGE;
int gender_choice = random() % 2;
if (gender_choice == 0 && male_count > 0)
   Asigna al usuario género "Masculino" y un nombre aleatorio de la lista de nombres masculinos si hay
      nombres disponibles y el género seleccionado es masculino.
 else if (female_count > 0)
      Asigna al usuario género "Femenino" y un nombre aleatorio de la lista de nombres femeninos si hay
      nombres disponibles.
generate_random_hobbies(user->hobbies, hobbies_list, hobby_count);
generate_random_personality(user->personality, personalities_list, personality_count);
```

Definition at line 70 of file user.c.

5.27.2.4 load_file()

```
void load_file (
            const char * filename,
            char file_array[MAX_FILE_LINES][MAX_NAME_LENGTH],
            int * count)
```

Carga un archivo de texto en un arreglo. Esta función lee un archivo de texto y almacena cada línea en un arreglo bidimensional, hasta un máximo de líneas especificado por MAX_FILE_LINES.

Carga un archivo de texto en un arreglo. Lee un archivo de texto y almacena cada línea en un arreglo bidimensional de cadenas.

Parameters

filename	Nombre del archivo a cargar.
file_array	Arreglo donde se almacenarán las líneas del archivo.
count	Puntero a la variable que almacena el número de líneas leídas.

Carga un archivo de texto en un arreglo.

```
 Nombre del archivo a cargar.
FILE *file = fopen(filename, "r");
if (!file) ->Error
*count = 0;
char line[MAX_NAME_LENGTH];
while (*count < MAX_FILE_LINES && fgets(line, sizeof(line), file))
    Lee líneas del archivo y las guarda en un arreglo hasta alcanzar el límite o el final del archivo.
fclose(file);
```

Definition at line 16 of file user.c.

5.27.2.5 print_users()

```
void print_users (
            const User * user)
```

Imprime la información de un usuario. Esta función imprime en la salida estándar (consola) los detalles de un usuario, incluyendo su ID, nombre, género, edad, personalidad y hobbies.

Imprime la información de un usuario. Imprime los detalles de un usuario, incluyendo su ID, nombre, género, edad, personalidad y hobbies en la salida estándar.

Parameters

user	Puntero al usuario que se desea imprimir.

Imprime la información de un usuario.

```
if (!user) ->Error
fprintf(stdout, "ID: %d\n", user->id);
fprintf(stdout, "Nombre: %s\n", user->username);
fprintf(stdout, "Género: %s\n", user->gender);
fprintf(stdout, "Edad: %d\n", user->age);
fprintf(stdout, "Personalidad: %s\n", user->personality);
fprintf(stdout, "Hobbies:\n");
for (int i = 0; i < MAX_HOBBIES && user->hobbies[i][0] != '\0'; i++)
  Muestra el hobbie del usuario.
```

Definition at line 214 of file user.c.

5.28 user.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00016 void load_file(const char *filename, char file_array[MAX_FILE_LINES][MAX_NAME_LENGTH], int *count)
00017 {
00030     FILE *file = fopen(filename, "r");
00031     if (!file)
00032     {
00033         fprintf(stderr, "Error al abrir el archivo %s, saliendo...\n", filename);
00034         exit(EXIT_FAILURE);
00035     }
00036
00037     *count = 0;
00038     char line[MAX_NAME_LENGTH];
00039
00040     while (*count < MAX_FILE_LINES && fgets(line, sizeof(line), file))
00041     {
00042         size_t len = strcspn(line, "\n");
00043         line[len] = '\0';
00044
00045         if (len > MAX_NAME_LENGTH)
00046             len = MAX_NAME_LENGTH - 1;
00047
00048         memcpy(file_array[*count], line, len);
00049         file_array[*count][len] = '\0';
00050         (*count)++;
00051     }
00052
00053     fclose(file);
00054 }
00055
00070 void generate_random_users(User *user, int id, char male_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH],
      int male_count, char female_usernames[MAX_FILE_LINES][MAX_NAME_LENGTH], int female_count, char
      hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int hobby_count, char
      personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH], int personality_count)
00071 {
00087     if (!user)
00088     {
00089         fprintf(stderr, "No se puede generar un usuario nulo. Saliendo...\n");
00090         exit(EXIT_FAILURE);
00091     }
00092
00093     user->id = id;
00094
00095     for (int i = 0; i < MAX_HOBBIES; i++)
00096         user->hobbies[i][0] = '\0';
00097
00098     user->age = random() % MAX_AGE + MIN_AGE;
00099
00100     int gender_choice = random() % 2;
00101
00102     if (gender_choice == 0 && male_count > 0)
00103     {
00104         strncpy(user->gender, "Masculino", MAX_GENDER - 1);
00105         user->gender[MAX_GENDER - 1] = '\0';
00106         int name_index = random() % male_count;
```

```
00107         strncpy(user->username, male_usernames[name_index], MAX_NAME_LENGTH - 1);
00108         user->username[MAX_NAME_LENGTH - 1] = '\0';
00109     }
00110     else if (female_count > 0)
00111     {
00112         strncpy(user->gender, "Femenino", MAX_GENDER - 1);
00113         user->gender[MAX_GENDER - 1] = '\0';
00114         int name_index = random() % female_count;
00115         strncpy(user->username, female_usernames[name_index], MAX_NAME_LENGTH - 1);
00116         user->username[MAX_NAME_LENGTH - 1] = '\0';
00117     }
00118
00119     generate_random_hobbies(user->hobbies, hobbies_list, hobby_count);
00120     generate_random_personality(user->personality, personalities_list, personality_count);
00121 }
00122
00130 void generate_random_hobbies(char hobbies[MAX_HOBBIES][MAX_HOBBIE_LENGTH], char
      hobbies_list[MAX_FILE_LINES][MAX_HOBBIE_LENGTH], int hobby_count)
00131 {
00145     if (hobby_count <= 0)
00146     {
00147         fprintf(stderr, "No hay hobbies disponibles. Saliendo...\n");
00148         exit(EXIT_FAILURE);
00149     }
00150
00151     for (int i = 0; i < MAX_HOBBIES; i++)
00152         hobbies[i][0] = '\0';
00153
00154     int *hobbie_selected = calloc(hobby_count, sizeof(int));
00155
00156     if (!hobbie_selected)
00157         return;
00158
00159     int num_hobbies = (random() % MAX_HOBBIES) + 1;
00160     int added_hobbies = 0;
00161
00162     for (int i = 0; i < num_hobbies && added_hobbies < MAX_HOBBIES; i++)
00163     {
00164         int hobbie_index = random() % hobby_count;
00165
00166         if (!hobbie_selected[hobbie_index])
00167         {
00168             strncpy(hobbies[added_hobbies], hobbies_list[hobbie_index], MAX_HOBBIE_LENGTH - 1);
00169             hobbies[added_hobbies][MAX_HOBBIE_LENGTH - 1] = '\0';
00170             hobbie_selected[hobbie_index] = 1;
00171             added_hobbies++;
00172         }
00173     }
00174
00175     free(hobbie_selected);
00176 }
00177
00186 void generate_random_personality(char *personality, char
      personalities_list[MAX_FILE_LINES][MAX_PERS_LENGTH], int personality_count)
00187 {
00197     if (personality_count <= 0)
00198     {
00199         fprintf(stderr, "No hay personalidades disponibles. Saliendo...\n");
00200         exit(EXIT_FAILURE);
00201     }
00202
00203     personality[0] = '\0';
00204
00205     strcpy(personality, personalities_list[random() % personality_count]);
00206 }
00207
00214 void print_users(const User *user)
00215 {
00230     if (!user)
00231     {
00232         fprintf(stderr, "No se puede imprimir un usuario nulo. Saliendo...\n");
00233         exit(EXIT_FAILURE);
00234     }
00235
00236     fprintf(stdout, "ID: %d\n", user->id);
00237     fprintf(stdout, "Nombre: %s\n", user->username);
00238     fprintf(stdout, "Género: %s\n", user->gender);
00239     fprintf(stdout, "Edad: %d\n", user->age);
00240     fprintf(stdout, "Personalidad: %s\n", user->personality);
00241     fprintf(stdout, "Hobbies:\n");
00242
00243     for (int i = 0; i < MAX_HOBBIES && user->hobbies[i][0] != '\0'; i++)
00244         fprintf(stdout, " - %s\n", user->hobbies[i]);
00245 }
```

5.29 src/user_log.c File Reference

Contiene funciones para interactuar con el archivo de log de usuarios, permitiendo contar, cargar, limpiar y agregar usuarios al historial.

```
#include "main.h"
```

Functions

- void user_count_from_log (int *user_count)

  Cuenta el número de usuarios en el archivo de log. Esta función lee el archivo "users_log.txt" y cuenta cuántos usuarios están registrados.
- int log_check ()

  Verifica si el archivo de log contiene registros. Esta función abre el archivo de log y verifica si contiene datos, devolviendo 1 si el archivo no está vacío y 0 si está vacío.
- void log_input (User users[ ])

  Lee el archivo de log y carga los datos de los usuarios en un arreglo. Esta función carga la información de los usuarios (ID, nombre, género, edad, personalidad, hobbies) desde el archivo "users_log.txt" y los guarda en el arreglo de usuarios proporcionado.
- void log_clean ()

  Limpia el archivo de log de usuarios. Esta función borra todo el contenido del archivo "users_log.txt" abriéndolo en modo de escritura sin agregar nuevos datos, efectivamente vaciando el archivo.
- void log_output (const User *user)

  Agrega un nuevo usuario al archivo de log. Esta función recibe un usuario y lo agrega al archivo "users_log.txt", registrando su ID, nombre, género, edad, personalidad y hobbies.

5.29.1 Detailed Description

Contiene funciones para interactuar con el archivo de log de usuarios, permitiendo contar, cargar, limpiar y agregar usuarios al historial.

Date

08-12-2024

Authors

Miguel Loaiza, Felipe Paillacar, Ignacio Contreras, Benjamín Sanhueza y Johann Fink

Definition in file user_log.c.

5.29.2 Function Documentation

5.29.2.1 log_check()

```
int log_check ()
```

Verifica si el archivo de log contiene registros. Esta función abre el archivo de log y verifica si contiene datos, devolviendo 1 si el archivo no está vacío y 0 si está vacío.

Función que verifica el estado del archivo de logs.

Returns

1 si hay historial, 0 si no lo hay.

Verifica si el archivo de log contiene registros.

```
FILE *file = fopen("./input/users_log.txt", "r");
if (file == NULL) -> Error
int char_file = fgetc(file);
fclose(file);
return (char_file != EOF);
```

Definition at line 51 of file user_log.c.

5.29.2.2 log_clean()

```
void log_clean ()
```

Limpia el archivo de log de usuarios. Esta función borra todo el contenido del archivo "users_log.txt" abriéndolo en modo de escritura sin agregar nuevos datos, efectivamente vaciando el archivo.

Función que limpia el archivo de logs, eliminando todos los registros de usuarios. Limpia el archivo de log de usuarios.

```
FILE *file = fopen("./input/users_log.txt", "w");
if (file == NULL) -> Error
fclose(file);
```

Definition at line 144 of file user_log.c.

5.29.2.3 log_input()

```
void log_input (
            User users[])
```

Lee el archivo de log y carga los datos de los usuarios en un arreglo. Esta función carga la información de los usuarios (ID, nombre, género, edad, personalidad, hobbies) desde el archivo "users_log.txt" y los guarda en el arreglo de usuarios proporcionado.

Parameters

users	Arreglo de usuarios donde se almacenarán los datos leídos del archivo.

Lee el archivo de log y carga los datos de los usuarios en un arreglo.

```
char line[256];
int user_count = 0;
int hobby_count = 0;
FILE *file = fopen("./input/users_log.txt", "r");
if (!file) -> Error
while (fgets(line, sizeof(line), file) && user_count < MAX_USERS)
     Detecta el comienzo de la línea de texto para cada caso
fclose(file);
```

Definition at line 83 of file user_log.c.

5.29.2.4 log_output()

```
void log_output (
            const User * user)
```

Agrega un nuevo usuario al archivo de log. Esta función recibe un usuario y lo agrega al archivo "users_log.txt", registrando su ID, nombre, género, edad, personalidad y hobbies.

Función que imprime la información de un usuario desde el log.

Parameters

user	Puntero al usuario que se desea agregar al archivo de log.

Función que agrega un nuevo usuario al archivo de log.

```
FILE *file = fopen("./input/users_log.txt", "a");
if (!file) -> Error
fprintf(file, "ID: %d\n", user->id);
fprintf(file, "Nombre: %s\n", user->username);
fprintf(file, "Género: %s\n", user->
fprintf(file, "Edad: %d\n", user->age);
fprintf(file, "Personalidad: %s\n", user->personality);
for (int i = 0; i < MAX_HOBBIES && user->hobbies[i][0] != '\0'; i++)
    fprintf(file, " - %s\n", user->hobbies[i]);
fprintf(file, "---\n");
fclose(file);
```

Definition at line 170 of file user_log.c.

5.29.2.5 user_count_from_log()

```
void user_count_from_log (
            int * user_count)
```

Cuenta el número de usuarios en el archivo de log. Esta función lee el archivo "users_log.txt" y cuenta cuántos usuarios están registrados.

Función que cuenta el número de usuarios registrados en el archivo de logs.

Parameters

user_count	Puntero a la variable que almacenará el número de usuarios encontrados.

Cuenta el número de usuarios en el archivo de log.

```
FILE *file = fopen("./input/users_log.txt", "r");
const char *key_word = "ID:";
char buffer[100];
if (file == NULL) -> Error
while (fscanf(file, "%99s", buffer) == 1)
    if (strcasecmp(buffer, key_word) == 0)
       (*user_count)++;
fclose(file);
```

Definition at line 14 of file user_log.c.

5.30 user_log.c

Go to the documentation of this file.

```
00001
00007 #include "main.h"
00008
00014 void user_count_from_log(int *user_count)
00015 {
00029     FILE *file = fopen("./input/users_log.txt", "r");
00030     const char *key_word = "ID:";
00031     char buffer[100];
00032
00033     if (file == NULL)
00034     {
00035         fprintf(stderr, "No se puede acceder al historial para el conteo de usuarios. Saliendo...");
00036         exit(EXIT_FAILURE);
00037     }
00038
00039     while (fscanf(file, "%99s", buffer) == 1)
00040         if (strcasecmp(buffer, key_word) == 0)
00041             (*user_count)++;
00042
00043     fclose(file);
00044 }
00045
00051 int log_check()
00052 {
00063     FILE *file = fopen("./input/users_log.txt", "r");
00064     if (file == NULL)
00065     {
00066         fprintf(stderr, "No se puede acceder al historial para su checkeo. Saliendo...");
00067         exit(EXIT_FAILURE);
00068     }
00069
00070     int char_file = fgetc(file);
00071
00072     fclose(file);
00073
00074     return (char_file != EOF);
00075 }
00076
00083 void log_input(User users[])
00084 {
00098     char line[256];
00099     int user_count = 0;
00100     int hobby_count = 0;
00101
00102     FILE *file = fopen("./input/users_log.txt", "r");
00103     if (!file)
00104     {
00105         fprintf(stderr, "No se pudo abrir el archivos del historial para el ingreso de usuarios.
      Saliendo...");
00106         exit(EXIT_FAILURE);
00107     }
00108
00109     while (fgets(line, sizeof(line), file) && user_count < MAX_USERS)
00110     {
00111         if (strncmp(line, "ID:", 3) == 0)
00112         {
00113             sscanf(line, "ID: %d", &users[user_count].id);
00114             hobby_count = 0;
00115         }
00116         else if (strncmp(line, "Nombre:", 7) == 0)
00117             sscanf(line, "Nombre: %12[^\n]", users[user_count].username);
00118         else if (strncmp(line, "Género:", 7) == 0)
00119             sscanf(line, "Género: %12[^\n]", users[user_count].gender);
00120         else if (strncmp(line, "Edad:", 5) == 0)
00121             sscanf(line, "Edad: %d", &users[user_count].age);
00122         else if (strncmp(line, "Personalidad:", 13) == 0)
00123             sscanf(line, "Personalidad: %6[^\n]", users[user_count].personality);
00124         else if (strncmp(line, " - ", 3) == 0)
00125         {
00126             if (hobby_count < MAX_HOBBIES)
00127             {
00128                 sscanf(line, " - %50[^\n]", users[user_count].hobbies[hobby_count]);
00129                 hobby_count++;
00130             }
00131         }
00132         else if (strncmp(line, "---", 3) == 0)
00133             user_count++;
00134     }
00135
00136     fclose(file);
00137 }
00138
```

```
00144 void log_clean()
00145 {
00154     FILE *file = fopen("./input/users_log.txt", "w");
00155 
00156     if (file == NULL)
00157     {
00158         fprintf(stderr, "Error al abrir el archivo para limpieza del historial. Saliendo...");
00159         exit(EXIT_FAILURE);
00160     }
00161     fclose(file);
00162 }
00163 
00170 void log_output(const User *user)
00171 {
00188     FILE *file = fopen("./input/users_log.txt", "a");
00189     if (!file)
00190     {
00191         fprintf(stderr, "Error al crear el historial de usuarios. Saliendo...\n");
00192         exit(EXIT_FAILURE);
00193     }
00194 
00195     fprintf(file, "ID: %d\n", user->id);
00196     fprintf(file, "Nombre: %s\n", user->username);
00197     fprintf(file, "Género: %s\n", user->gender);
00198     fprintf(file, "Edad: %d\n", user->age);
00199     fprintf(file, "Personalidad: %s\n", user->personality);
00200 
00201     for (int i = 0; i < MAX_HOBBIES && user->hobbies[i][0] != '\0'; i++)
00202         fprintf(file, " - %s\n", user->hobbies[i]);
00203 
00204     fprintf(file, "---\n");
00205     fclose(file);
00206 }
```

Index