

ESTRUCTURAS DE DATOS Y ALGORITMOS II, 2022-2023

Memòria projecte: FitFeet

Pau Noguera Mulet U213916

Jan Prats Soler U213927

Xavier Riera U214387

Adriana Salinas U215119

11/06/2023

LLISTA DE CONTINGUTS:

INTRODUCCIÓ	2
OBJECTIUS DEL PROJECTE	3
Objectius obligatoris aconseguits	3
Objectius desitjables aconseguits	4
Objectius exploratoris adquirits	5
SOLUCIÓ	6
Arquitectura del sistema	6
Gestió d'errors	8
Disseny del model de dades	9
Descripció i processament del conjunt de dades	10
Menu.c:	11
Menu.h:	25
SOCIAL.c:	27
USUARI.c:	31
USUARI.h:	46
REFERÈNCIES	47

INTRODUCCIÓ

Per al projecte de pràctiques de l'assignatura, Estructura de dades i algoritmes II, se'ns ha proposat desenvolupar un prototip de Xarxa Social funcional utilitzant els nostres coneixements de programació adquirits durant el curs. Aquesta serà una aplicació local, per consola i en llenguatge C. Per tant, un projecte complicat, però a la mateixa vegada, molt interessant i que ens permet aprendre i posar en pràctica els conceptes adquirits.

Després d'uns dies parlant i debatent, vam decidir tematitzar la nostra xarxa social com una plataforma dedicada al món del 'running', sorgint així el nom "FitFeet". Amb aquesta decisió presa, varem començar a elaborar el nostre projecte i seguir els objectius de cada sessió, tot i que inicialment sempre suposa una certa dificultat, fins i tot quan disposem d'una guia ben estructurada. A poc a poc, vam anar constraint la xarxa i sorgiren diversos problemes que no ens permeteren fer totes les funcionalitats que volíem implementar. No obstant això, ens sentim satisfets dels èxits i el resultat obtingut en el desenvolupament d'una xarxa social funcional, gràcies a la dedicació i esforç col·lectiu.

En aquest informe, documentarem el procés de desenvolupament del projecte FitFeet, des de l'explicació de totes les funcions implementades, l'arquitectura i disseny de la xarxa, les estructures de dades utilitzades, la gestió d'errors i exposar els objectius aconseguits.

OBJECTIUS DEL PROJECTE

Objectius obligatoris aconseguits

Els objectius obligatoris del projecte s'han complert satisfactoriament. A continuació es descriurà com s'han implementat cada un d'aquests dins de l'estructura i s'explicaran en més profunditat posteriorment.

En primer lloc, s'han implementat una llista dinàmica utilitzant l'estructura `user_list`. Aquesta llista enllaçada d'usuaris permet emmagatzemar i gestionar eficientment la informació dels usuaris. D'altre lloc, hem implementat una cua en l'estructura “Queue”, la qual s'utilitza per gestionar totes aquelles tasques relacionades amb els usuaris de la xarxa social. La cua s'encarregarà del seguiment de l'usuari inicial amb el camp “head” i l'usuari final amb el camp “last”. Finalment, hem implementat una pila amb l'estructura de dades “PilaPublicacions”, aquesta pila s'utilitza per emmagatzemar totes les publicacions d'un usuari, i així aquestes es mostren les més recents primer.

Cal remarcar que hem aplicat un algoritme de cerca com en aquest cas el “Linear Search”. Aquesta s'ha aplicat repetides vegades com per exemple a l'hora de buscar la direcció de memòria d'un usuari en concret en la llista d'usuaris. D'altra banda, hem dissenyat un “Quick Sort” per tal d'ordenar la pila de paraules més utilitzades a menys utilitzades.

A més, hem dissenyat una estructura de dades que és un diccionari per tal d'emmagatzemar totes les paraules de totes les publicacions enregistrades pels usuaris.

Finalment, des del primer dia vàrem començar amb el github. Encara que vàrem tenir molts de problemes a l'hora d'activar-lo per primera cop, però una vegada hi vàrem poder accedir-hi tots, ens ha facilitat molt la feina en grup i la creació del codi. Això sí, a vegades els commits and push no funcionaven i des d'un altre dispositiu quan volies fer un upgrade del codi no l'actualitzava correctament. La solució que nosaltres hem trobat és la de fer un clone del projecte, aleshores, sí que s'actualitzen correctament els canvis.

Objectius desitjables aconseguits

Per tal de provar totes les funcionalitats del codi durant el procés de fer el projecte, és més còmode tenir usuaris ja creats guardats dins un fitxer. Així, els llegim i es creen automàticament els usuaris.

Per aconseguir-ho, hem creat un arxiu de text (PERFIL.txt) amb la següent informació:

```
Pau 123 Noguera Mulet 21 del@gmail.com |Llucmajor basquet futbol formula1 ciclisme pingpong 0 1 94
Ariana 123 Salinas 1d 43 fel@gmail.com Tiana Voleibol Rugby Ciclisme Gimnastica Esgrima 0 0 156
Jan 123 Prats Pons 4 jdf@gmail.com Maresme Handbol Golf Patinatge Taekwondo Badminton 0 0 612
Xavi 123 Riera Pons 23 gkm@gmail.com Montuiri rugby futbol castellers natacio basquet 0 0 0
Maria 123 Noguera Mulet 34 fekmd@gmail.com Llucmajor Natacio Badminton ioga running basquet 0 0 0
```

Cada dada és separada per un espai (' ').

Aleshores, en la funció `llegir_usuaris_desde_arxiu`, accedim al fitxer PERFIL.txt. I, així, llegim les dades d'una font externa (arxiu de text) i s'agilitza el procés.

Per una altra banda, alhora de fer la nostra xarxa més especial entre les altres, l'hem tematitzat i donat una capa de personalització a nivell conceptual.

Aleshores, la nostra Xarxa Social està dirigida a persones que lis agrada anar a córrer, és a dir, per a runners. I el primer dia varem elegir el nom de FitFeet (per votació), on altres propostes eren: RunMate o Km-unity. Una altre modificació seria la dels 5 gustos, en el nostre cas, són 5 esports preferits.

Llavors, hem afegit una dada personal més, la dels quilòmetres recorreguts per cada usuari (persona). Aquesta dada es demana un cop feta la publicació. Per tant, està pensat que cada vegada que hagis anat a correr, primer facis una publicació de text i després introduesquis els quilòmetres fets aquell dia. El nombre de quilòmetres també es guarda en l'arxiu de text i cada vegada es suma el nou nombre de quilòmetres fets. I aquest es queda guardat per sempre (fins a modificar-lo). Gràcies a això, podem fer un rànquing dels usuaris amb més quilòmetres recorreguts.

Objectius exploratoris adquirits

Al principi ens varem plantejar fer una capa estètica afegint alguna biblioteca i implementar les seves interfícies, però varem trobar que seria una feina bastant complicada, per això, ens hem limitat a fer un menús agradables a la vista i comprensibles.

```
void menu() {
    printf( format: "\n");
    printf( format: "===== Benvingut a Feetfit =====\n");
    printf( format: "| 1. Insereix nou usuari |\n");
    printf( format: "| 2. Llistar tots els usuaris existents |\n");
    printf( format: "| 3. Iniciar sessió |\n");
    printf( format: "| 4. Sortir |\n");
    printf( format: "===== Selecciona una opció: ");
}

void printf_menu_usuari() {
    printf( format: "\n");
    printf( format: "---| 1. Perfil |---\n");
    printf( format: "---| 2. Enviar sol·licituds d'amistat |---\n");
    printf( format: "---| 3. Acceptar/denegar sol·licituds |---\n");
    printf( format: "---| 4. Llista d'amics |---\n");
    printf( format: "---| 5. Realitzar una publicació |---\n");
    printf( format: "---| 6. Llistar les publicacions |---\n");
    printf( format: "---| 7. Llistar paraules TOP |---\n");
    printf( format: "---| 8. Ranking km recorreguts |---\n");
    printf( format: "---| 9. Sortir |---\n");
}
```

Addicionalment, com hem comentat abans, hem fet ús d'arxius de text per emmagatzemar informació i facilitar el procés. Però, aquest arxiu també ens dóna la possibilitat de guardar tant les dades personals, com les sol·licituds o amistats que un usuari pot tenir. I així, cada vegada que sortim de la xarxa social, aquesta informació no es perd.

La implementació de les funcionalitats de lectura i guardat de dades desde un arxiu extern estan aplicades en les funcions: `guardar_usuaris_en_arxiu`, `llegir_usuaris_desde_arxiu` i `emmagatzema_dades`.

SOLUCIÓ

Arquitectura del sistema

L'arquitectura de la nostra xarxa social, FitFeet, es basa en una estructura modular composta per diversos components interconnectats. Principalment, tenim els arxius .c (on es defineixen les funcions) i els arxius .h (on es declaren les funcions), a més, estan els arxius de text, on s'hi emmagatzema informació de cada usuari. D'aquesta manera, els blocs principals són: menú, social i usuari.

Menú:

El menú està compost per l'arxiu menu.c i menu.h, on tenim totes les funcions relacionades amb el menú, i aquestes estan dividides en tres apartats: menú, opcions de menú i auxiliars.

- Menú: codi de les definicions de les funcions menu, select_option i print_option.
- Opcions de menú: les diferents funcionalitats dins el menú com afegir_usuari, print_users, menu_usuari, printf_menu_usuari.
- Auxiliars: funcions poden reutilitzar en diferents parts del programa com comprovar_correu, comprovar_usuari, resp_bol, checkPassword, inicialitzarQueue, isQueueEmpty, enqueue, dequeue.

A més, en el menu.h estan les funcions declarades i totes les estructures de dades necessàries per a tot el projecte, aquestes són: Paraula, st_Diccionari, Publicacio, PilaPublicacions, User, user_list, Queue.

Social:

En el bloc social hi formen part les funcionalitats dedicades a la interacció entre usuaris (les sol·licituds d'amistat), aquestes funcions són: enviar_solicitud, acceptar_denegar_solicituds i llistar_amics_acceptats, definides en SOCIAL.c i declarades en SOCIAL.h.

Usuari:

En el bloc usuari, tenim les funcions dedicades als usuaris (definides en USUARI.c i declarades en SOCIAL.h), dividides en tres apartats: funcions elementals, publicació i diccionari.

- Funcions elementals: funcionalitats fonamentals que serveixen per ser la base del funcionament de l'usuari dins la nostra xarxa social, com: guardar_usuaris_en_arxiu, llegir_usuaris_desde_arxiu, emmagatzema_dades, print_user_info, canvi_de_dades.
- Publicació: funcions dirigides a fer possible a l'usuari a la realització de publicacions: fer_publicacio, Timeline, quilometres, print_rankingKM.
- Diccionari, algoritmes que s'encarreguen del funcionament dels diccionaris: swap, particio, quicksort, trending, buscar_pàrraula, afegir_pàrraula_nova.

Finalment, hem de tenir en compte els arxius externs de text, on es guarda informació necessària dels usuaris per tal de no perdre-la i conservar-la una vegada s'ha sortit de la xarxa. Aquests arxius .txt són: perfil, amics i solicituds_amics:

En el PERFIL, s'emmagatzema totes les dades personals de cada usuari en una línia de text diferents, així, s'hi pot accedir i tenir sempre els mateixos perfils d'usuaris. En AMICS, a cada línia està escrit el nom de cada usuari, i seguidament, el noms dels seus amics. I en SOLICITUD_AMICS, també com en AMICS, a cada línia després del nom de l'usuari estan el noms de qui els ha enviat una sol·licitud d'amistat, que una vegada s'ha acceptat la sol·licitud, s'elimina i passa a AMICS.

Gestió d'errors

Durant el nostre procés del projecte hem tingut diferents errors, però al final els hem acabat gestionant correctament per tal que el codi acabes compilant i funcionant amb normalitat.

El primer error que vam tenir va ser a l'hora de connectar-mos tots els membres de l'equip al Github. No sabíem que es feia i, d'aquesta manera, vàrem estar tota la primera sessió intentant compartir el projecte. Malgrat això, ho aconseguirem i ens ha facilitat molt la feina en grup.

Un altre error seria en el tot el tema de sol·licituds, vam definir malament el current i el head, i com a conseqüència no s'enviaven correctament les sol·licituds i feia que no poguessim avançar en el projecte. Dies més tard ens vam adonar que la clau estava en fer una funció amb 2 inputs en comptes de 1. S'havien de definir 2 variables de tipus user: current i iterar_llista, aquesta primera seria l'usuari actual seleccionat a la llista i iterar_llista que s'inicialitza amb el valor del punter head de una estructura prèviament definida “user_list”. “Llista és una linked list i head és un punter al primer element de la llista.

```

8
9      // ----- SOL·LICITUDS D'AMISTAT -----
10  ↗ int enviar_solicitud(user_list* Llista, User *usuari) {
11      User* current = usuari;
12      User* iterar_llista = Llista -> head;
13      char receptor[MAX_LENGTH];
    ↗

```

Un cop vam poder gestionar aquest problema ja vam poder implementar altres funcionalitats respecte al tema de sol·licituds.

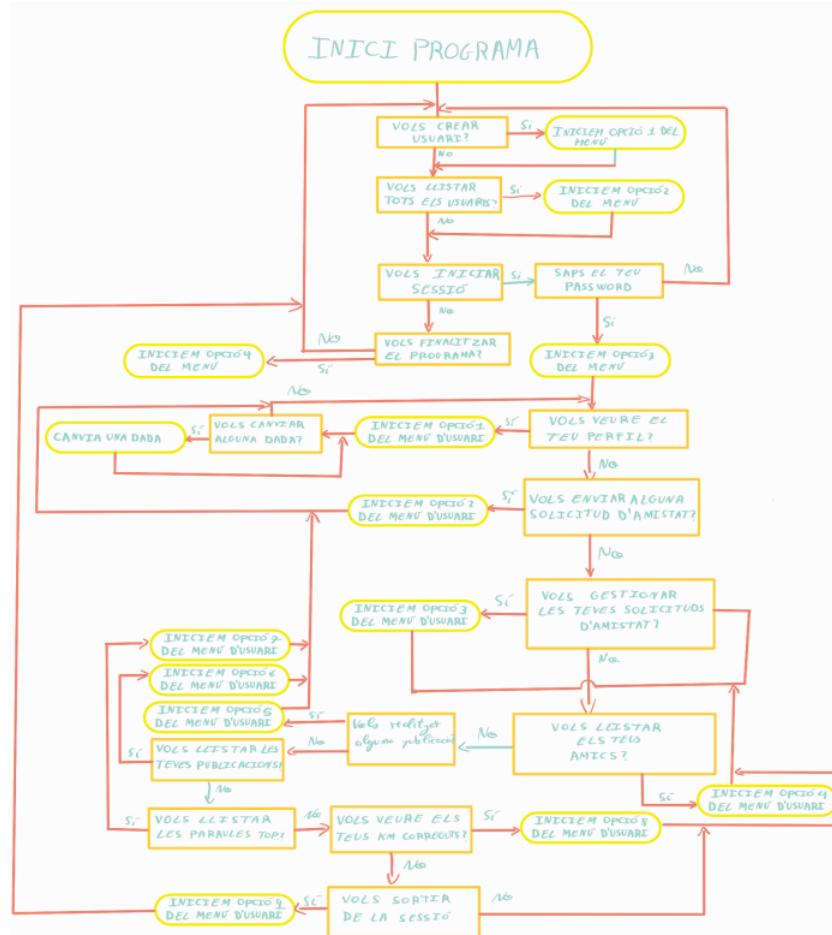
A més a més, si alguna vegada qualcú dels membres de l'equip no li sortia la funció que estava realitzant, entre tots, intentavem ajudar i solucionar els dubtes que tenia. Per tant, en feina en equip i solidaritat hem tret endavant la nostra Xarxa Social.

Addicionalment, hem fet ús del Chat GPT per a la revisió d'algunes funcions que ens han costat més fer, per veure si algunes funcions es podien millorar o tenir un millor gestionament d'errors. Per exemple, en la funció *quilometres*, ens ha dit que havíem de comprovar que la resposta introduïda fos un nombre enter i no un char.

Disseny del model de dades

Es pot observar en el diagrama de flux dos bucles infinitos principals. El primer faria referència al menú principal de la nostre xarxa social. En aquest l'usuari podrà anar escollit entre quatre opcions diferents, i pel contrari se'l farà torna a escollir entre una d'aquestes quatre de forma perpètua. Evidentment, una de les opcions a escollir és la de finalitzar programa, la qual és la única opció en tot el programa que ens permetrà finalitzar-lo.

En segon lloc, es pot observar com a partir d'una ramificació en comencen a derivar tot un segon bloc. Aquest fa referència al menú de l'usuari, és a dir, a totes les funcionalitats de l'usuari. També cal destacar que està tancat en un bucle infinit a excepció que l'usuari triï l'opció 9, la qual el farà sortir de la seva sessió d'usuari.



Descripció i processament del conjunt de dades

Nosaltres vam decidir fer servir estructures per tal de poder ser més òptims i més organitzats, les estructures les definiriem i explicarem posteriorment (apartat [menu.h](#)) però és important saber que sense les estructures el codi seria una mica “caòtic” ja que les estructures en C ens permeten crear variables noves del tipus que sigui l’estructura per tal de poder treballar amb elles de forma fàcil i òptima.

L'exemple més clar d'això és la struct User que conté la nostra xarxa social. Ja que es poden crear moltes variables de tipus User que continguin tot el que té l'estructura interna de User, que ara vindria a ser doncs, el nom d'usuari, l'edat, el correu electrònic, l'ubicació... Tanmateix si ho féssim sense estructures, no podríem crear una variable local de tipus User, ja que cada cop que volguessim fer servir un usuari, haurièm de emmagatzemar les seves dades en variables noves cada vegada fent que augmenti exponencialment la seva complexitat, també tindriem el problema de tenir més probabilitat d'errors, ja que com més tipus diferents de variables fem servir més probabilitat hi ha de confondre o de fer servir alguna variable que no toca.

Per tant en el nostre codi hem hagut de fer servir moltes estructures, ja que la creació d'una xarxa social és prou complexe i es necessiten estructures per tal de poder organitzar el codi i fer-lo el més òptim possible.

Primer de tot, per un tema principal d'organització hem dividit el nostre codi en diferents arxius .c i .h, on en els .c definim el codi de les funcions, i en els .h posem les estructures i les declaracions de les funcions.

En total tenim tres arxius .c (menu, SOCIAL i USUARI) i els seu pertinents arxius .h. A més de tres arxius de text (PERFIL, AMICS i SOLICITUDS_AMICS).

Menu.c:

En el fitxer menu.c tenim totes les funcions relacionades amb el menú, aquestes estan dividides en tres apartats: menú, opcions de menú i auxiliars.

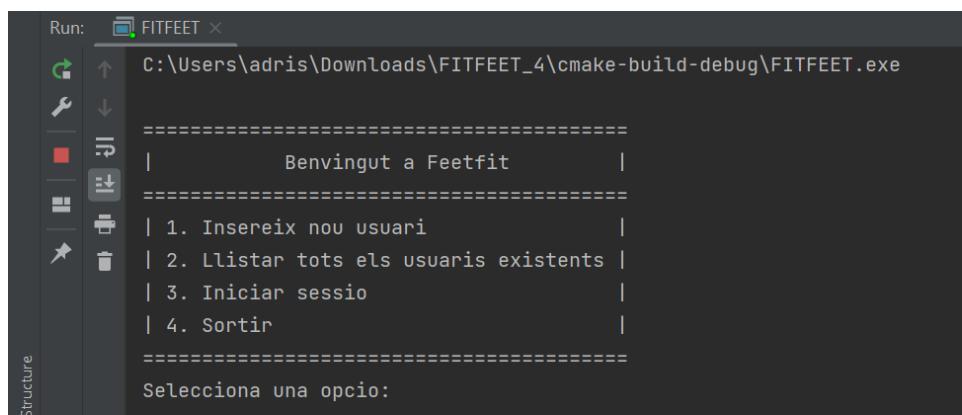
Per començar, abans de fer cap estructura ni res, vam inicialitzar el nostre codi fent una funció que imprimeix per pantalla el menú principal de la nostre xarxa social, d'aquesta manera podem organitzar d'una manera més pràctica com ramificarem el nostre codi:

```

12
13     /// ----- MENÚ -----
14     void menu() {
15         printf("\n");
16         printf("===== Benvingut a Feetfit =====\n");
17         printf("|\t\t\t Benvingut a Feetfit\t |\n");
18         printf("===== \n");
19         printf("|\t 1. Insereix nou usuari\t\t |\n");
20         printf("|\t 2. Llistar tots els usuaris existents\t |\n");
21         printf("|\t 3. Iniciar sessió\t\t\t |\n");
22         printf("|\t 4. Sortir\t\t\t\t\t |\n");
23         printf("===== \n");
24         printf("Selecciona una opció: ");
25     }
26

```

De tal manera que quan compilem el codi i l'executem tenim aquest resultat:



```

Run: FITFEET ×
C:\Users\adris\Downloads\FITFEET_4\cmake-build-debug\FITFEET.exe
=====
|      Benvingut a Feetfit      |
=====
| 1. Insereix nou usuari      |
| 2. Llistar tots els usuaris existents |
| 3. Iniciar sessió          |
| 4. Sortir                   |
=====
Selecciona una opció:

```

Seguidament per tal de poder accedir a cadascuna d'aquestes opcions del nostre menú necessitem una funció que ens porti a accedir dins de cadascuna de les opcions. Declarem una variable de tipus enter, per tal de poder emmagatzemar el número d'opció que ens posa l'usuari. Fem un bucle infinit perquè si es posa malament el número es pugui tornar a posar sense que ens fagin fora.

```

26
27 int select_option() {
28     int option; // variable per guardar el valor introduït per l'usuari
29     while (1) { // entra en un bucle infinit
30         if (scanf("%d", &option) != 1) { // intenta llegir un enter i comprova si s'ha llegit correctament
31             while (getchar() != '\n'); // si la lectura no ha sigut correcte, esborra el contingut del buffer?
32         } else if (option >= 1 && option <= 4) { // comprova que 1 i 4 són opcions vàlides
33             printf("\n");
34             printf("\n");
35             return option;
36         }
37         printf("Opció incorrecta. Torna a intentar.\n");
38     }
39 }
```

Per tal de poder accedir a cadascuna d'aquestes opcions hem de posar a cada opció la funció corresponent que farà el que posa al menú: però com la majoria son funcions que encara no hem vist, així per resumir és una funció on hi han 4 if on cadascun entra a una opció del menú principal:

```

40
41 int print_option(int option, user_list *Llista, st_Diccionari* TaulaHash) {
42     if (option == 1) { //opció inserir nou usuari.
43         User *usuari = (User *) malloc(sizeof(User)); // crea un nou usuari amb memòria dinàmica
44         emmagatzema_dades(usuari, Llista);
45         afegir_usuari(Llista, usuari);
46         return 0;
47     }
48     } else if (option == 2) { //opció imprimir llista d'usuaris.
49         print_users(Llista);
50         sleep(1.5);
51         return 0;
52     }
53     } else if (option == 3) { //opció per interactuar amb les opcions d'un usuari.
54         menu_usuari(Llista, TaulaHash);
55         return 0;
56     }
57     } else if (option == 4) { //Finalitzar amb el programa.
58         return 1;
59     }
60 }
```

Un cop que ja poguem inserir la nostre opció i que el programa la pugui llegir; ara ens hem de centrar en la part de registrar un nou usuari a la nostre xarxa social: Un cop definida la struct User i user_list (comentades més avall en l'informe) és el moment que en una struct de tipus User vagi guardant cada usuari registrat; ho farem de la següent forma: farem una llista d'usuaris registrats: una linked list, que tindrà el seu head i el seu next.

```

63 // ----- OPCIONS DE MENÚ -----
64 void afegir_usuari(user_list* llista, User* usuari) {
65     if (llista->head == NULL) { // si la llista està buida...
66         llista->head = usuari; // el primer element de la llista és el usuari nou
67     } else {
68         User* temp = llista->head; // si la llista no està buida...
69         while (temp->next != NULL) { // temp apunta al head
70             temp = temp->next; // mentre no s'ha arribat al final de la llista...
71         }
72         temp->next = usuari; // temp avança al següent element de la llista (fins que arriba al final)
73     }
74 }
75

```

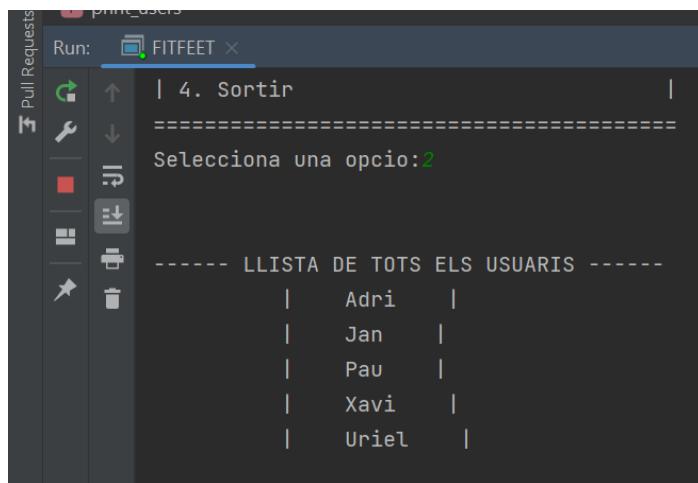
Ja podem anar guardant els diferents usuaris registrats a la nostre xarxa social; però clar d'una forma o altre hem de poder saber quins usuaris s'han registrat, per això tenim una funció prou important com és la funció print_users(), com bé indica el seu nom, l'objectiu principal d'aquesta funció és imprimir aquests usuaris registrats, per fer-ho hem implementat una cua (funció que també veurem definida més avall en l'informe) però el cas és que ha de sortir quan li donem a la opció 2 del nostre menú principal, i també ha d'aparèixer aquesta llista d'usuaris registrats un cop vulguis iniciar sessió:

```

87
88 void print_users(user_list* llista) {
89     printf("----- LLISTA DE TOTS ELS USUARIS -----\\n");
90
91     Queue queue;
92     inicialitzarQueue(&queue);
93
94     User* current = llista->head; // el current és el primer de la llista
95     while (current != NULL) { // si la llista no està buida, entrem al bucle
96         enqueue(&queue, current->nom); // fica el usuari current dins la cua
97         current = current->next; // el current és el següent de la llista
98     }
99
100    while(!isEmpty(&queue)) { // mentre la cua no estigui buida, entrem al bucle, per escriure cada nom dels usuaris
101        printf(" | %s | \\n", queue.head->nom);
102        dequeue(&queue);
103    }
104

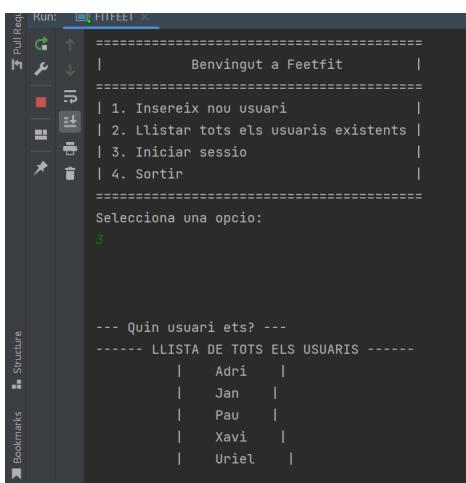
```

Aquí veuriem un resultat de com s'haurien de veure els nostres usuaris registrats en la nostra xarxa social si seleccionessim la opció numero 2 del menú principal:



```
Run: FITFEET
| 4. Sortir
=====
Selecciona una opcio:2
-----
----- LLISTA DE TOTS ELS USUARIS -----
| Adri   |
| Jan    |
| Pau    |
| Xavi   |
| Uriel  |
```

Seguidament, quan seleccionem la opció 3 d'iniciar sessió també ens apareix la llista de tots els usuaris registrats, seguidament preguntarà que amb qui usuari vol iniciar sessió:

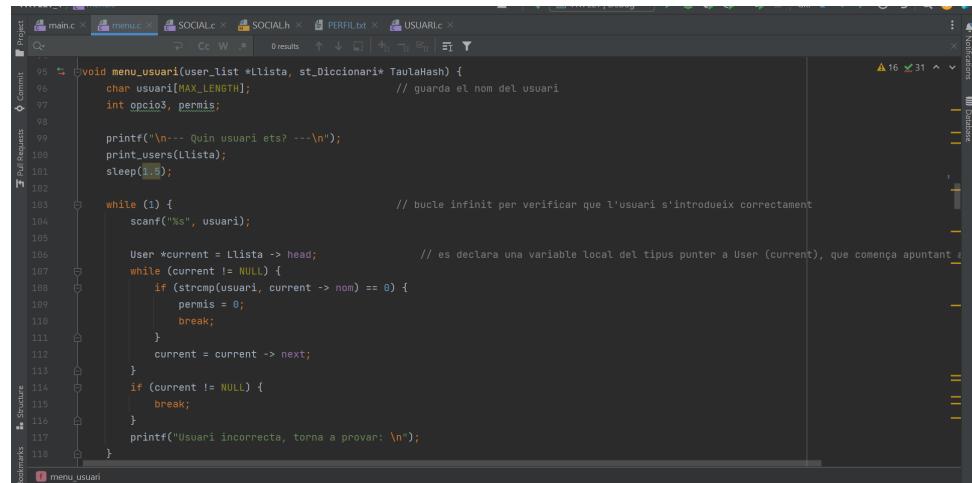


```
Run: FITFEET
=====
|     Benvingut a Feetfit      |
=====
| 1. Insereix nou usuari      |
| 2. Llistar tots els usuaris existents |
| 3. Iniciar sessio            |
| 4. Sortir                   |
=====
Selecciona una opcio:
3

--- Quin usuari ets? ---
----- LLISTA DE TOTS ELS USUARIS -----
| Adri   |
| Jan    |
| Pau    |
| Xavi   |
| Uriel  |
```

A continuació tenim una funció prou important com és la funció menu_usuari, que s'encarrega de mostrar el submenú d'iniciar sessió, és a dir, és el menú que apareixerà quan estiguis dintre d'un perfil registrat.

La funció comença preguntant qui usuari registrat ets, on prèviament haurà llistat tots els usuaris registrats, i d'allà tu escrius amb el que vulguis iniciar sessió. Això ho farà implementant la funció print_users(Llista) definida més a dalt. Fem un scanf perquè la persona pugui introduir un usuari, i després fem un petit algoritme que busqui l'usuari introduït amb l'usuari registrat, i quan coincideixin, pararà el bucle.

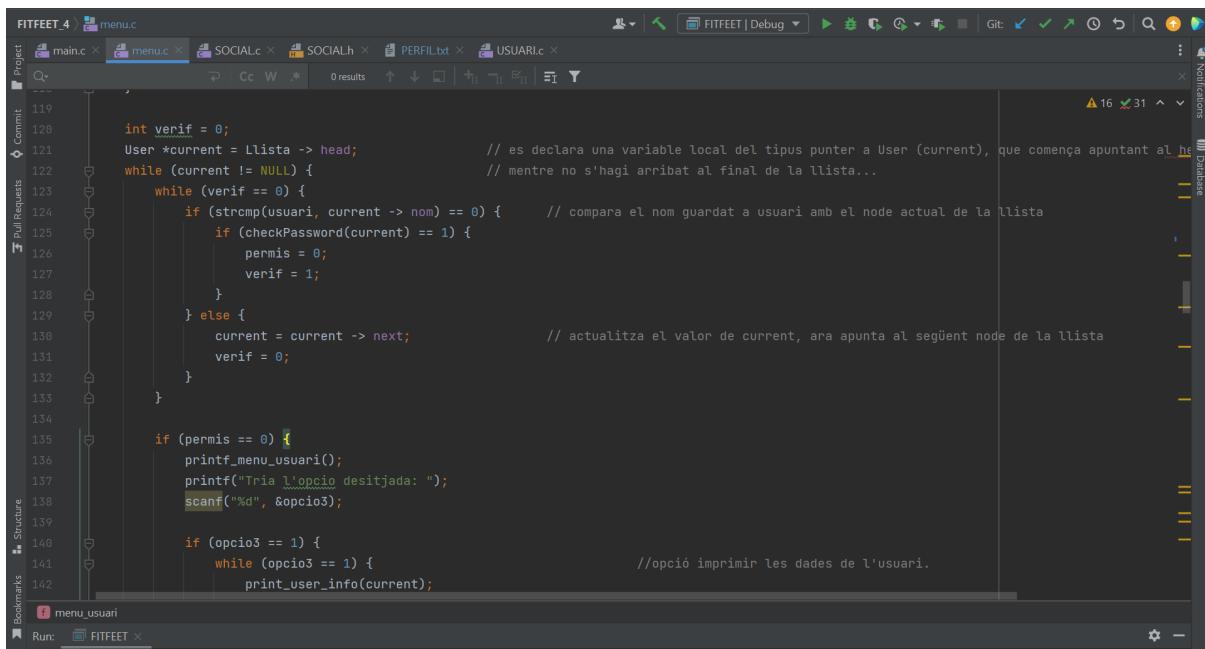


```

95 void menu_usuari(user_list *llista, st_Diccionari* TaulaHash) {
96     char usuari[MAX_LENGTH]; // guarda el nom del usuari
97     int opcio3, permis;
98
99     printf("\n--- Quin usuari ets? ---\n");
100    print_users(llista);
101    sleep(.5);
102
103    while (1) { // bucle infinit per verificar que l'usuari s'introdueix correctament
104        scanf("%s", usuari);
105
106        User *current = Llista -> head; // es declara una variable local del tipus punter a User (current), que comença apuntant al head de la llista
107        while (current != NULL) {
108            if (strcmp(usuari, current -> nom) == 0) {
109                permis = 0;
110                break;
111            }
112            current = current -> next;
113        }
114        if (current != NULL) {
115            break;
116        }
117        printf("Usuari incorrecte, torna a provar: \n");
118    }
119
120    int verif = 0;
121    User *current = Llista -> head; // es declara una variable local del tipus punter a User (current), que comença apuntant al head de la llista
122    while (current != NULL) { // mentre no s'hagi arribat al final de la llista...
123        while (verif == 0) {
124            if (strcmp(usuari, current -> nom) == 0) { // compara el nom guardat a usuari amb el node actual de la llista
125                if (checkPassword(current) == 1) {
126                    permis = 0;
127                    verif = 1;
128                }
129            } else {
130                current = current -> next; // actualitza el valor de current, ara apunta al següent node de la llista
131                verif = 0;
132            }
133        }
134
135        if (permis == 0) {
136            printf_menu_usuari();
137            printf("Tria l'opció desitjada: ");
138            scanf("%d", &opcio3);
139
140            if (opcio3 == 1) {
141                while (opcio3 == 1) {
142                    print_user_info(current); //opció imprimir les dades de l'usuari.
143                }
144            }
145        }
146    }
147
148    if (permis == 1) {
149        printf("Has fet un error en la teva acció. Torna a provar.\n");
150    }
151
152    sleep(.5);
153}

```

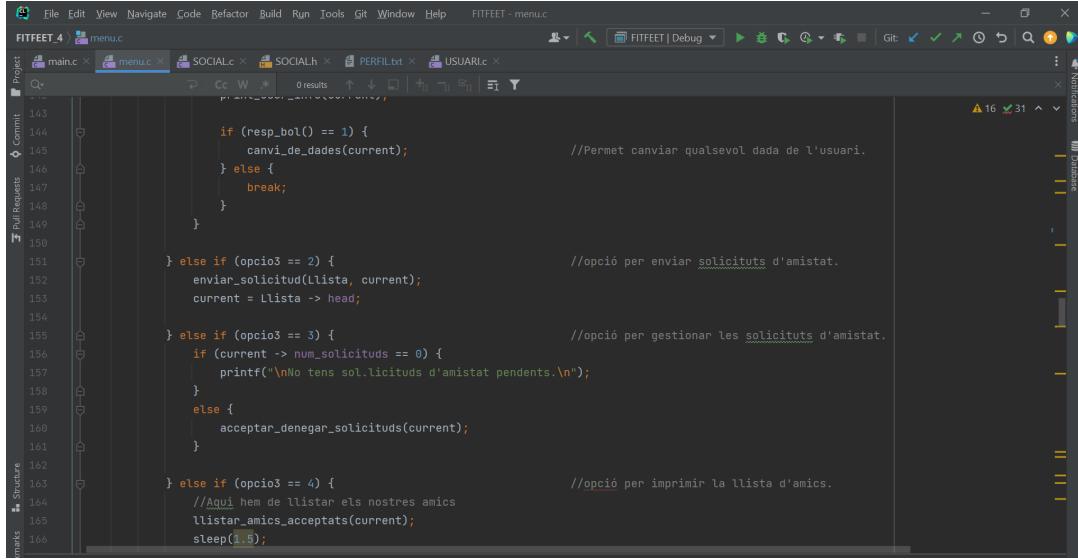
La segona part de la funció `menu_usuari()` consisteix en que hem creat una variable anomenada `current` de tipus `User` que ara és el `head` de la llista, llavors aquest petit algoritme mira i comprova que l'usuari amb que s'ha iniciat sessió sigui el `current` de la llista, osigui que sigui el `head`. Seguidament fem servir la funció `printf_menu_usuari()` que la definim una mica més abaix; i ens fa triar entre 9 opcions: on cadascuna té la seva funció associada:



```

119
120    int verif = 0;
121    User *current = Llista -> head; // es declara una variable local del tipus punter a User (current), que comença apuntant al head de la llista
122    while (current != NULL) { // mentre no s'hagi arribat al final de la llista...
123        while (verif == 0) {
124            if (strcmp(usuari, current -> nom) == 0) { // compara el nom guardat a usuari amb el node actual de la llista
125                if (checkPassword(current) == 1) {
126                    permis = 0;
127                    verif = 1;
128                }
129            } else {
130                current = current -> next; // actualitza el valor de current, ara apunta al següent node de la llista
131                verif = 0;
132            }
133        }
134
135        if (permis == 0) {
136            printf_menu_usuari();
137            printf("Tria l'opció desitjada: ");
138            scanf("%d", &opcio3);
139
140            if (opcio3 == 1) {
141                while (opcio3 == 1) {
142                    print_user_info(current); //opció imprimir les dades de l'usuari.
143                }
144            }
145        }
146    }
147
148    if (permis == 1) {
149        printf("Has fet un error en la teva acció. Torna a provar.\n");
150    }
151
152    sleep(.5);
153}

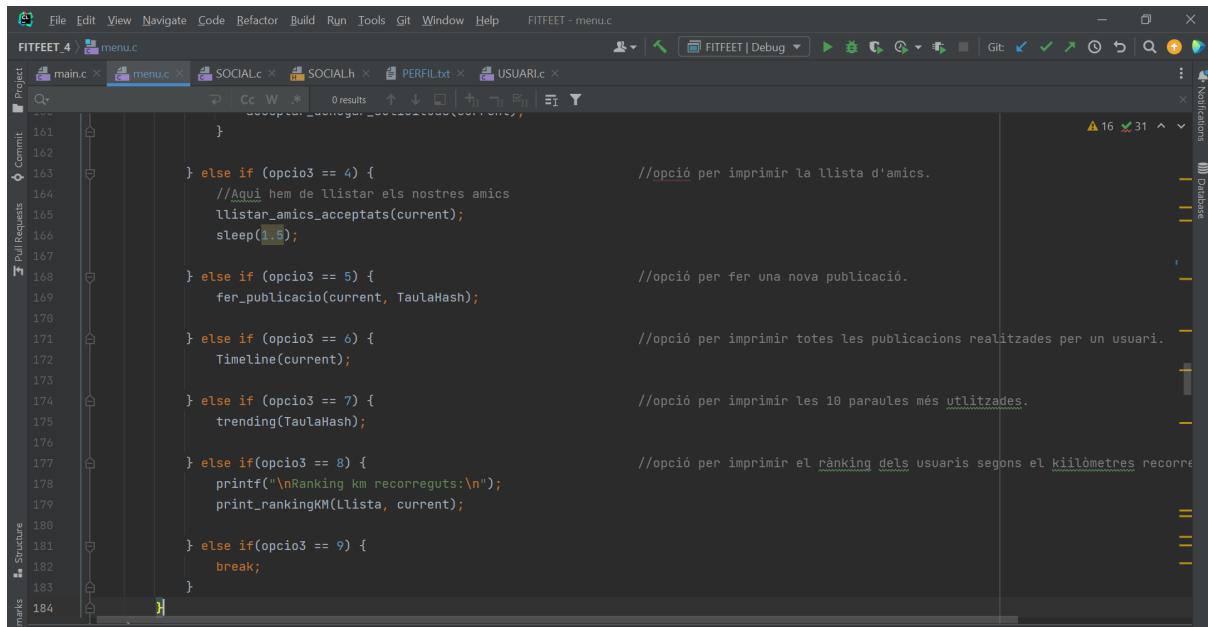
```



```

143     if (resp_bol() == 1) {
144         canvi_de_dades(current); //Permet canviar qualsevol dada de l'usuari.
145     } else {
146         break;
147     }
148 }
149
150
151 } else if (opcion3 == 2) { //opció per enviar solicitudes d'amistat.
152     enviar_solicitud(llista, current);
153     current = Llista -> head;
154
155 } else if (opcion3 == 3) { //opció per gestionar les solicitudes d'amistat.
156     if (current -> num_solicitudes == 0) {
157         printf("\nNo tens sol.licituds d'amistat pendents.\n");
158     }
159     else {
160         acceptar_denegar_solicitudes(current);
161     }
162
163 } else if (opcion3 == 4) { //Aquí hem de llistar els nostres amics
164     llistar_amics_aceptats(current);
165     sleep(1.5);
166
167 }
168
169 } else if (opcion3 == 5) { //opció per imprimir la llista d'amics.
170     //Aquí hem de llistar els nostres amics
171     llistar_amics_aceptats(current);
172     sleep(1.5);
173
174 } else if (opcion3 == 6) { //opció per fer una nova publicació.
175     Timeline(current);
176
177 } else if (opcion3 == 7) { //opció per imprimir totes les publicacions realitzades per un usuari.
178     trending(TaulaHash);
179
180 } else if (opcion3 == 8) { //opció per imprimir el rànking dels usuaris segons el km recorreguts
181     printf("\nRanking km recorreguts:\n");
182     print_rankingKM(llista, current);
183
184 } else if (opcion3 == 9) {
185     break;
186 }
187
188 void printf_menu_usuari() {
189     printf("\n");
190     printf(" ---| 1. Perfil |---\n");
191     printf(" ---| 2. Enviar sol.solicituds d'amistat |---\n");
192     printf(" ---| 3. Acceptar/denegar sol.solicituds |---\n");
193     printf(" ---| 4. Llista d'amics |---\n");
194     printf(" ---| 5. Realitzar una publicacio |---\n");
195     printf(" ---| 6. Llistar les publicacions |---\n");
196     printf(" ---| 7. Llistar paraules TOP |---\n");
197     printf(" ---| 8. Ranking km recorreguts |---\n");
198     printf(" ---| 9. Sortir |---\n");
199 }

```



```

161
162
163 } else if (opcion3 == 4) { //opció per imprimir la llista d'amics.
164     //Aquí hem de llistar els nostres amics
165     llistar_amics_aceptats(current);
166     sleep(1.5);
167
168 } else if (opcion3 == 5) { //opció per fer una nova publicació.
169     fer_publicacio(current, TaulaHash);
170
171 } else if (opcion3 == 6) { //opció per imprimir totes les publicacions realitzades per un usuari.
172     Timeline(current);
173
174 } else if (opcion3 == 7) { //opció per imprimir les 10 paraules més utilitzades.
175     trending(TaulaHash);
176
177 } else if (opcion3 == 8) { //opció per imprimir el rànking dels usuaris segons el km recorreguts
178     printf("\nRanking km recorreguts:\n");
179     print_rankingKM(llista, current);
180
181 } else if (opcion3 == 9) {
182     break;
183 }
184
185 void printf_menu_usuari() {
186     printf("\n");
187     printf(" ---| 1. Perfil |---\n");
188     printf(" ---| 2. Enviar sol.solicituds d'amistat |---\n");
189     printf(" ---| 3. Acceptar/denegar sol.solicituds |---\n");
190     printf(" ---| 4. Llista d'amics |---\n");
191     printf(" ---| 5. Realitzar una publicacio |---\n");
192     printf(" ---| 6. Llistar les publicacions |---\n");
193     printf(" ---| 7. Llistar paraules TOP |---\n");
194     printf(" ---| 8. Ranking km recorreguts |---\n");
195     printf(" ---| 9. Sortir |---\n");
196 }

```

La funció print_menu_usuari és per imprimir per pantalla el submenú de l'inici de sessió:



```

187
188 void printf_menu_usuari() {
189     printf("\n");
190     printf(" ---| 1. Perfil |---\n");
191     printf(" ---| 2. Enviar sol.solicituds d'amistat |---\n");
192     printf(" ---| 3. Acceptar/denegar sol.solicituds |---\n");
193     printf(" ---| 4. Llista d'amics |---\n");
194     printf(" ---| 5. Realitzar una publicacio |---\n");
195     printf(" ---| 6. Llistar les publicacions |---\n");
196     printf(" ---| 7. Llistar paraules TOP |---\n");
197     printf(" ---| 8. Ranking km recorreguts |---\n");
198     printf(" ---| 9. Sortir |---\n");
199 }

```

En el cas de les funcions auxiliars, són aquelles funcions que serveixen per facilitar tasques específiques del programa, que es poden reutilitzar en diferents parts del programa.

Aquestes són: *comprovar_correu*, *comproavar_usuari*, *resp_bol*, *checkPassword*, *inicialitzarQueue*, *isQueueEmpty*, *enqueue*, *dequeue*. A més, totes estan declarades en el fitxer menu.h.

```
/// ----- AUXILIARS -----
bool comprovar_correu(char *correu);
bool comproavar_usuari(user_list* llista, char* nom);
int resp_bol();
int checkPassword(User *usuari);
void inicialitzarQueue(Queue* queue);
int isQueueEmpty(Queue* queue);
void enqueue(Queue* queue, char* nom);
void dequeue(Queue* queue);
```

Comprovar correu:

La funció *comprovar_correu*, com el seu nom indica, té la tasca de comprovar que el correu introduït per consola quan es crea un nou usuari sigui vàlid.

Es crida la funció amb el punter de la variable de caràcters correu `bool comprovar_correu(char *correu)` com a una funció booleana, és a dir, retorna true o false. Per tant, comprovarà si el correu és correcta (true) o no (false).

Primer es defineixen i inicialitzen quatre variables:

- *longitud*: és el nombre enter de la longitud del correu
- *posArroba*: és el nombre enter de la posició on es troba '@'
- *teArroba*: variable booleana per indicar si hi ha arroba, inicialitzada com a *false*
- *tePunt*: variable booleana per indicar si hi ha punt, inicialitzada com a *false*

```
int longitud = strlen( Str: correu );
int posArroba = 0;
bool teArroba = false;
bool tePunt = false;
```

Després, comprovam que el correu no té una longitud menor a 5 caràcters, ja que si fos així, no seria un correu vàlid (hi ha d'haver un '@' + '.') i com a mínim una lletra entre els dos símbols i una lletra al principi i final = total mínim 5 caràcters).

```
if (longitud < 5) {
    return false;
}
```

Per tal que sigui un correu vàlid també hem de comprovar que només tingui un sol '@', aleshores, recorrem cada caràcter iterant i cercant '@'. Fins que el troba, i defineix *teArroba* com a true i guarda la posició ('i') del '@'. Segueix iterant els caràcters per veure si hi ha un altre '@', si el troba retorna que el correu és invàlid (false). Sinó, surt del for i comprovem que si no s'ha trobat '@' retorna false (correu incorrecta).

```

for (int i = 0; i < longitud; i++) {
    if (correo[i] == '@') {
        if (teArroba) {
            return false;
        } else {
            teArroba = true;
            posArroba = i;
        }
    }
}

if (!teArroba) {
    return false;
}
  
```

Finalment, comprovem que només té un '.' i aquest està després del '@'. Una altra vegada iterem amb el for cada caràcter desde el de després del '@' fins al darrer. Si trobem el '.' canviem la variable *tePunt* a true i sortim del for. Si no es troba el punt, el correu és invàlid (retorna false). I si no hi ha hagut cap problema retorna true (correu vàlid).

```

for (int i = posArroba + 1; i < longitud; i++) {
    if (correo[i] == '.') {
        tePunt = true;
        break;
    }
}

if (!tePunt) {
    return false;
}

return true;
  
```

Aquesta funció, està definida en el fitxer menu.c entre les línies 203 i 243 i en el menu.h línia 84. I s'utilitza (declara) en la línia 236 del fitxer USUARI.c per a la funció *emmagatzema_dades*. On després de demanar per consola el correu de l'usuari que estem creant, entre a un bucle infinit per comprovar que el correu és vàlid, i no surt del bucle fins que s'ha introduït un correu correcta.

```

printf( format: "Introdueix el teu correu electrònic: \n");      //Es demana el correu electrònic de l'usuari el qual es validarà a continuació.
scanf( format: "%s", usuari -> correo);

while (1) {
    if (comprovar_correu(usuari -> correo) == true) {           //funció per validar que el correu està ben escrit
        printf( format: "El correu es valid.\n\n");
        break;
    } else {
        printf( format: "El correu es invalid. Torna a introduir el teu correu: \n");
        scanf( format: "%s", usuari -> correo);
    }
}
  
```

Una de les limitacions de l'algoritme és que suposa que la cadena del correu (introduïda per consola) no té espais. Per tant, una millora seria solucionar-ho. El temps emprat per implementar aquesta funció ha estat de 20 minuts.

Comprovar_usuari:

La funció *comprovar_usuari* té com a finalitat comprova si un determinat usuari es troba en la llista d'usuaris. Es crida la funció amb el punter de la llista d'usuaris *llista* i un punter al nom introduït `bool comprovar_usuari(user_list* llista, char* nom)` com a funció booleana, és a dir, retorna true (si l'usuari es troba a la llista) o false (si l'usuari no es troba a la llista). La funció recorre la llista d'usuaris buscant una coincidència amb el nom d'usuari especificat.

Les variables utilitzades són:

- *llista*: punter a la llista d'usuaris
- *nom*: cadena de caràcters del nom de l'usuari (escrit per consola)
- *temp*: punter temporal per recórrer la llista

Primer, comprovem que la llista no està buida.

Després, la variable temporal apunta al primer element (usuari) de la llista (*head*) i mentres la llista no estigui buida (no arribi al final de la llista), s'entra al bucle per recorrer la llista i comparar (amb *strcmp*) el nom actual (del *temp*) amb el nom d'usuari específic. Si coincideix, la funció retorna *true*. En cas contrari, s'avança al següent de la llista actualitzant *temp* a *next*. Finalement, si no s'han trobat cap coincidència, es retorna *false* (l'usuari no es troba a la llista).

```
bool comprovar_usuari(user_list* llista, char* nom) {
    if (llista -> head == NULL) {
        return false;
    }

    User* temp = llista -> head;
    while (temp != NULL) {
        if (strcmp (temp -> nom, nom) == 0) {
            return true;
        }
        temp = temp -> next;
    }
    return false;
}
```

Aquesta funció està definida en el fitxer menu.c entre les línies 245 i 258 i en el menu.h en la línia 85. I s'utilitza en la línia 19 del fitxer SOCIAL.c per a la funció *enviar_solicitud* on es comprova mitjançant un bucle while que l'usuari a qui vols enviar la sol·licitud d'amistat és un usuari existent. I també s'usa en la línia 213 del fitxer USUARI.c en la funció *emmagatzema_dades* per a comprovar per mitjà d'un bucle while que el nom del nou usuari que estem afegint, existent anteriorment en un altre usuari.

```
while(1) {
    printf( format: "Introdueix el teu nom: \n");
    scanf( format: "%s", usuari -> nom);

    if (comprovar_usuari( llista, usuari -> nom)) {
        printf( format: "L'usuari ja existeix. \n");
    } else {
        break;
    }
}
```

Resp_bol:

La funció *resp_bol* demana a l'usuari si vol modificar alguna opció de les dades personals una vegada estas dins el teu perfil d'usuari. Si la resposta és 'si', es retorna el valor 1 per realitzar les modificacions, si s'escriu (per consola) 'no', es retorna 0 per no modificar res. Per tant, es crida la funció `int resp_bol()` com a funció que retorna un enter.

Primer s'inicialitzen les variables:

- *resposta*: array de 10 caràcters com a màxim que guarda la resposta escrita per consola de l'usuari.
- *respostaValida*: variable booleana definida com a *false* que indica si la resposta és vàlida o no.

Entrem dins el bucle, ja que la resposta és invàlida (*respostaValida == false*), aleshores, es llegeix per consola i es converteix, amb un for, la resposta a minúscules amb la funció `tolower`. I després comprovem si la resposta coincideix amb 'si' (retorna 1) o 'no' (retorna 0), amb la funció `strcmp`. Però si la resposta no coincideix amb cap de les opcions vàlides, es mostra un missatge d'error i continua demanant a l'usuari una resposta vàlida.

```
int resp_bol() {
    char resposta[10];
    bool respostaValida = false;

    printf("¿Vols modificar alguna opción? (sí/no): ");

    while (!respostaValida) { // bucle infinit
        scanf("%s", resposta);

        for (size_t i = 0; i < strlen(resposta); i++) { // converteix la resposta a minúscules
            resposta[i] = tolower(resposta[i]);
        }

        if (strcmp(resposta, "sí") == 0) { // si la resposta es 'sí'
            respostaValida = true; // la resposta es válida
            return 1; // i retornem 1 (per modificar alguna opción)
        } else if (strcmp(resposta, "no") == 0) { // si la resposta es 'no'
            respostaValida = true; // la resposta es válida
            return 0; // i retornem 0 (per no modificar res)
        } else {
            printf("Resposta inválida. Introduceix una respuesta correcta ('sí', 'no').\n"); // resposta inválida
        }
    }
}
```

Aquesta funció està definida en el fitxer menu.c entre les línies 260 i 285 i en el menu.h en la línia 86. I s'utilitza en la línia 144 del propi fitxer menu.c per a la funció *menu_usuari* on una vegada s'ha iniciat sessió de l'usuari correctament, i hem entrat dins el nostre perfil del submenú, ens demana si volem modificar alguna de les nostres dades personals. En el cas de que s'hagi respondut que 'sí' (1) s'inicia la funció *canvi_de_dades*, però si la resposta és 'no' (0) es surt del submenú: perfil.

```
if (opcion3 == 1) {
    while (opcion3 == 1) {
        print_user_info(current);

        if (resp_bol() == 1) {
            cambi_de_dades(current);
        } else {
            break;
        }
    }
}
```

Una possible limitació seria l'overflow en l'entrada de la resposta de l'usuari.

CheckPassword:

La funció `checkPassword` comprova si la contrasenya introduïda per l'usuari coincideix amb la contrasenya guardada en l'estructura de l'usuari. Es crida la funció amb el punter de l'estructura d'usuari `int checkPassword(User *usuari)` que retorna un enter, si la contrasenya és correcta retorna 1, si és incorrecta retorna 0.

Les variables utilitzades són:

- *usuari*: punter a l'estructura `User` que conté les dades de l'usuari
- *input*: array de 20 caràcters (màx) que guarda la contrasenya introduïda per consola
- *correct*: variable booleana definida com a *false*

S'entra al bucle `while` infinit, ja que *correct* és fals, on es demana introduir la contrasenya de l'usuari (amb `scanf`). Aquest *input* és comparat amb la contrasenya emmagatzemada en l'estructura del usuari utilitzant la funció `strcmp`. Si les contrasenyes coincideixen, es mostra un missatge de benvinguda amb el nom de l'usuari, es retorna el valor 1 i finalitza la funció. Si les contrasenyes no coincideixen, es mostra un missatge d'error, es retorna el valor 0 i es tornarà a demanar la contrasenya.

```
int checkPassword(User *usuari) {
    char input[20];
    bool correct = false;

    while (!correct) {
        printf("Introdueix la contrasenya: ");
        scanf("%s", input);

        if (strcmp(input, usuari -> password) == 0) {
            printf("Benvingut %s!\n", usuari -> nom);
            return 1; // entrem al perfil
        } else {
            printf("Contrasenya incorrecta. Intenta-ho de nou.\n");
            return 0; // no entrem al perfil
        }
    }
}
```

Aquesta funció està definida en el fitxer `menu.c` entre les línies 287 i 304 i en el `menu.h` en la línia 87. I s'utilitza en la línia 125 del fitxer `menu.c` per a la funció `menu_usuari`, on una vegada introduït el nom de l'usuari a iniciar sessió et demana per la contrasenya de l'usuari per poder entrar al submenú.

```
while (verif == 0) {
    if (strcmp(usuari, current -> nom) == 0) {
        if (checkPassword(usuari, current) == 1) {
            permis = 0;
            verif = 1;
        }
    } else {
        current = current -> next;
        verif = 0;
    }
}
```

Una possible millora seria no permetre que la resposta de la contrasenya per consola sigui buida, ja que així no seria possible comprovar que es correcta.

Ara entrem dins les funcions destinades a implementar una fila (Queue), però encara seguim dins les funcions auxiliars del `menu.c`.

IncialitzarQueue:

La funció *inicialitzarQueue* té l'objectiu inicialitzar una cua buida (estructura Queue), només la prepara per ser utilitzada en algoritmes futurs. Es crida la funció així: `void inicialitzarQueue(Queue* queue)`.

Les variables utilitzades són:

- *queue*: punter a l'estructura Queue
- *head*: primer element de l'estructura Queue
- *last*: darrer element de l'estructura Queue

```
void inicialitzarQueue(Queue* queue) {
    queue -> head = NULL;
    queue -> last = NULL;
}
```

Simplement inicialitza les variables *head* i *last* de l'estructura Queue com a nules, és a dir, indica que no hi ha cap element en la primera i última posició de la cua.

Aquesta funció està definida entre les línies 308 i 311 en el menu.c i en la línia 88 del menu.h. I s'utilitza en la línia 81 (menu.c) per a la funció *print_users*, on simplement inicialitza la cua per a poder utilitzar les funcions de la cua i imprimir els usuaris en una llista.

IsQueueEmpty:

La funció *isQueueEmpty* comprova si la cua està buida o no. Es crida amb el punter de l'estructura Queue, `int isQueueEmpty(Queue* queue)` i retorna un enter, si la cua està buida (1) o no (0). Això és gràcies a verificar si el punter del *head* apunta a un valor *NULL*.

Les variables utilitzades són:

- *queue*: punter a l'estructura Queue
- *head*: primer element de l'estructura Queue

```
int isQueueEmpty(Queue* queue) {
    return (queue -> head == NULL);
}
```

L'algoritme retorna el resultat de si el punter *head* apunta a un valor *NULL* o no, si apunta a un valor *NULL* retorna 1 i, per tant, la cua està buida.

Està definida entre les línies 313 i 315 en el menu.c i en la 89 del menu.h. I s'utilitza en la línia 89 (menu.c) en la funció *print_user* per comprovar que la cua no estigui buida i entrar en el bucle per escriure en una llista els noms dels usuaris. També en la línia 322 (menu.c) en la funció enqueue per comprovar si la cua està buida i afegir el primer element de la cua com el *head* i *last* de la cua. Una altre aplicació és en la funció *dequeue* en la línia 333 per si la cua està buida imprimir per consola que la cua està buida i, per tant, no es poden eliminar elements de la cua.

Enqueue:

La funció `enqueue` afegeix un element (en aquest cas un usuari) a la cua. Es crida amb el punter a la cua i el nom de l'usuari a afegir `void enqueue(Queue* queue, char* nom)`.

Les variables utilitzades són:

- `queue`: punter a l'estructura Queue
- `nom`: punter a la cadena de caràcter que conté el nom de l'usuari a afegir.
- `usuari_nou`: variable per guardar el nou usuari de la cua
- `head`: primer element de l'estructura Queue
- `last`: darrer element de l'estructura Queue

Primer es reserva memòria per a un nou usuari utilitzant `malloc` i s'assigna al punter `usuari_nou`. I es copia el nom al nom del nou usuari amb `strcpy`. A més, el següent `usuari_nou` és `NULL` (és a dir, l'`usuari_nou` afegit anteriorment és el darrer de la cua). Després, es comprova si la cua està buida amb la funció `isQueueEmpty`, en cas que ho sigui, l'`usuari_nou` és l'únic de la llista. En cas contrari, el nou usuari és el darrer de la cua.

```
void enqueue(Queue* queue, char* nom) {
    User* usuari_nou = (User*) malloc( Size: sizeof(User));
    strcpy( Dest: usuari_nou -> nom, Source: nom);
    usuari_nou -> next = NULL;

    if (isQueueEmpty(queue)) {
        queue -> head = usuari_nou;
        queue -> last = usuari_nou;
    } else {
        queue -> last -> next = usuari_nou;
        queue -> last = usuari_nou;
    }
}
```

La funció està definida entre les línies 317 i 330 del menu.c i a la línia 90 del menu.h. I s'utilitza en la línia 85 (menu.c) en la funció `print_users`, on una vegada inicialitzada la cua afegeix el `nom` de cada usuari de la *Llista*, si aquesta no està buida. I al final els imprimeix en forma de llista.

Dequeue:

La funció `dequeue` elimina el primer usuari de la cua. Es crida amb el punter de la cua `Queue` `void dequeue(Queue* queue)`.

Les variables utilitzades són:

- `queue`: punter a l'estructura `Queue`
- `temporal`: variable temporal per guardar el primer usuari de la llista i eliminar-lo
- `head`: primer element de l'estructura `Queue`

Primer comprovem si la cua està buida, en aquest cas, imprimeix per pantalla un missatge. Sinó, es guarda el primer usuari de la cua en la variable `temporal`. Després s'actualitza el punter `head` de la cua per a que sigui el següent de la llista. I finalment, s'allibera (amb `free`) la memòria reservada per a l'usuari eliminat (el primer).

```
void dequeue(Queue* queue) {
    if (isQueueEmpty(queue)) {
        printf( format: "La cua esta buida.\n");
        return;
    }

    User* temporal = queue -> head;
    queue -> head = queue -> head -> next;
    free( Memory: temporal);
}
```

La funció està definida entre les línies 332 i 341 (`menu.c`) i a la línia 91 (`menu.h`). I s'utilitza en la línia 91 del `menu.c` a la funció `print_users` per anar eliminant els noms dels usuaris de la cua una vegada ja s'han imprimiu per pantalla en la llista.

Menu.h:

Un cop arribat a aquest punt, ara ja si, podem introduir les nostres estructures principals les quals algunes s'hi ressaltaran les seves finalitats més endavant.

Una de les estructures de dades utilitzades és “Paraula”, que s'ha utilitzat per tal d'emmagatzemar-hi una paraula amb el seu comptador associat. Aquesta estructura té dos camps: “paraula”, el qual és un vector de caràcters que emmagatzema la paraula, i “cont”, que el definim com un enter el qual indica el nombre de vegades que s'ha trobar aquesta paraula en el conjunt de totes les publicacions realitzades pels usuaris.

Seguidament, trobem la estructura “st_Diccionari”, la qual representa un diccionari de paraules. Aquesta estructura conté un array de punters a l'estructura “Paraula”, anomenat “paraules”, que emmagatzema les paraules del diccionari, i un enter “num_paraules” que indicarà la suma del nombre de paraules emmagatzemades en el diccionari.

Per gestionar les publicacions de la nostre xarxa social, s'utilitza l'estructura de dades “Publicacio”. Aquesta, té dos camps. El primer, “text” és un vector de caràcters que emmagatzema el contingut de la publicació en format caràcters. En segon lloc, la variable “seguent”, serà un punter a l'estructura “Publicacio” que apunta a la següent publicació en una llista enllaçada. Implementarem l'estructura “PilaPublicacions” la qual es una pila de publicacions la qual únicament contindrà el camp “top”. Aquesta variable és un punter a l'element superior de la pila, és a dir, l'última publicació afegida.

Una de les estructures més importants, sinó la que més, és l'estructura “User”. Aquesta representarà un usuari de la xarxa social i conté diversos camps per emmagatzemar tota la informació personal com el nom, contrasenya, edat, correu electrònic, ubicació, gustos i quilòmetres recorreguts. També té camps relacionats amb les connexions socials, és a dir, la variable “next”, serà un punter a l'usuari següent en una llista enllaçada d'usuaris. “num_amics” indicarà el nombre d'amics de l'usuari (d'aquí a que sigui una variable de classe enter), i “amics”, una array de punters a l'estructura “User” que emmagatzema les direccions de memòria dels amics de l'usuari. La variable “num_solicitiuds”, indicarà el nombre de sol·licituds d'amistat pendents, i “solicitiuds”, serà una array de punters a l'estructura “User” la qual s'hi emmagatzemarà totes les sol·licituds pendents. Finalment, declarem una pila de publicacions anomenada “publicacio” per enregistrar totes les publicacions de l'usuari.

Amb la finalitat d'emmagatzemar tots els usuaris en una llista, declarem l'estructura “user_list”, la qual serà una llista dinàmica d'usuaris enllaçats. Aquesta estructura té un

camp “head” que apunta a l’usuari inicial de la llista, i un enter “num_persones” que indica el nombre d’usuaris emmagatzemats a la llista.

Finalment, utilitzem l’estructura “Queue” per gestionar les tasques relacionades amb els usuaris de la xarxa social. L’estructura representa una cua d’usuaris la qual es defineix per dos camps: El “head”, el qual és un punter a l’usuari inicial de la cua, i “last”, que serà un punter a l’usuari final de la cua.

En resum, hem dissenyat estructures de dades de classe llistes dinàmiques, piles, i cues per tal de gestionar eficientment la informació dels usuaris, així com les seves publicacions i les seves relacions socials.

SOCIAL.c:

Funcionalitats dedicades a la interacció entre usuaris, aquestes funcions són: enviar_solicitud, acceptar_denegar_solicitudes i llistar_amics_accepteds.

Aquesta part del codi, s'encarrega de gestionar les interaccions entre usuaris de la xarxa social. Les tres funcions de les quals consta ens permeten enviar sol·licituds d'amistat entre usuaris, acceptar i denegar-les i comprovar la nostra llista d'amics (veure la llista d'usuaris presents a l'array amics de cada usuari).

Enviar_sol·licitud:

Començarem explicant la funció enviar_sol·licitud. Aquesta funció pren com a paràmetres dos punters: el punter a la llista d'usuaris existents a la xarxa social i el punter a una struct d'usuari (és a dir, un punter a l'struct de l'usuari que actualment té la sessió iniciada).

Per poder gestionar les sol·licituds, primer necessitem inicialitzar tres variables locals: la variable current (és la mateixa que el paràmetre usuari de la funció), la variable iterar_llista (ens servirà per iterar per tota la llista d'usuaris) i la variable receptor (és a dir, l'usuari a qui volem enviar la sol·licitud).

La funció comença amb un bucle while en el qual se'ns demana introduir el nom de l'usuari a qui volem enviar la sol·licitud d'amistat. El bucle es repeteix indefinidament fins que introduïm el nom d'un usuari que està present a la llista d'usuaris existents (ho fa mitjançant la funció comprovar_usuari prèviament explicada). Posteriorment, mitjançant un bucle for que es repeteix tantes vegades com usuaris hi ha la llista d'usuaris existents, s'itera a través de tota la llista fins trobar un usuari el nom del qual coincideix amb el nom introduït per l'usuari i es guarda a l'usuari trobat a la nova variable receptor_user.

```

7  while(1) {
8      printf( format: "A qui vols enviar una sol·licitud?");
9      scanf( format: "%s", receptor->nom);
10
11     if (comprovar_usuari( llista: Llista, receptor->nom)) {
12         break;
13     } else {
14         printf( format: "\nUsuari incorrecte, escriu un usuari existent\n");
15     }
16
17
18     // Cerquem al receptor en la llista
19     int index = -1;
20     for (int i = 0; i < Llista -> num_persones; i++) {
21         if (strcmp(iterar_llista -> nom, receptor->nom) == 0) {
22             index = i;
23             break;
24         }
25         iterar_llista = iterar_llista -> next;
26     }
27
28     User* receptor_user = iterar_llista;

```

Seguidament, es duen a terme set comprovacions mitjançant condicions if i bucles for per comprovar que no hi ha cap error. Les comprovacions que es realitzen són:

- comprovar que les variables current i receptor_user estan inicialitzades.
- comprovar que l'usuari a les variable current i receptor_user no són el mateix.
- s'itera per la llista d'amics de l'usuari current per comprovar que no el té d'amic a la llista.
- s'itera per la llista d'amics de l'usuari receptor_user per comprovar que no el té d'amic a la llista.
- s'itera per la llista de sol·licituds de l'usuari current per comprovar que no el té a la llista.
- s'itera per la llista de sol·licituds de l'usuari receptor_user per comprovar que no el té a la llista.
- es comprova que l'usuari receptor_user existeix a la llista d'usuaris existents.

```

    // Mirem que els paràmetres no estiguin buits
    if (current == NULL || receptor_user == NULL) {
        return -1;
    }

    // Comprovam que l'emissor i el receptor no siguin els mateixos
    if (current == receptor_user) {
        printf(format: "No et pots enviar una sol·licitud a tu mateix\n");
        return -1;
    }

    // Mirem si l'emissor ja té al receptor com amic
    for (int i = 0; i < current -> num_amics; i++) {
        if (current -> amics[i] == receptor_user) {
            printf(format: "Ja sou amics\n");
            return -1;
        }
    }

    // Mirem si el receptor ja té a l'emissor com amic
    for (int i = 0; i < receptor_user -> num_amics; i++) {
        if (receptor_user -> amics[i] == current) {
            printf(format: "Ja sou amics\n");
            return -1;
        }
    }
}

```

Finalment, si s'han passat totes les comprovacions, s'afageix a l'array de sol·licituds d'amistat de l'usuari receptor_user l'struct d'usuari de l'usuari current.

Acceptar_denyar_solicituds:

Aquesta funció serveix per accedir a la llista de sol·licituds d'amistat de l'usuari que té la sessió iniciada i gestionar les sol·licituds (es poden acceptar o es poden denegar). Pren com a paràmetres l'struct de l'usuari que té la sessió iniciada.

La funció comença comprovant que la llista de sol·licituds no està buida. Si no ho està, imprimeix un submenú mitjançant un bucle for en el qual apareixen els noms de tots els usuaris present a la llista de sol·licituds. Posteriorment, una vegada s'ha seleccionat quina sol·licitud d'amistat volem gestionar, s'inicialitza una nova variable usuari sol·licitant. Si s'accepta la sol·licitud, el nombre d'amics de la usuari i del sol·licitant augmenten en un, s'afegeixen l'un a l'altre a les respectives llistes d'amics i s'eliminen les sol·licituds d'amistat.

```

while(1) {
    printf( format: "Posa el numero que vulguis accionar: ");
    scanf( format: "%d", &opcion2);

    if (opcion2 == 1) {
        if (receptor->num_amics < MAX_AMICS && solicitant->num_amics < MAX_AMICS) {
            receptor->amics[receptor->num_amics] = solicitant;
            receptor->num_amics++;
            solicitant->amics[solicitant->num_amics] = receptor;
            solicitant->num_amics++;
            printf( format: "\nSol·licitud acceptada. Ara ets amic de %.\\n", solicitant->nom);

            // Remove the accepted request from the current user's request list
            for (int i = opcion - 1; i < receptor->num_solicitudes - 1; i++) {
                receptor->solicitudes[i] = receptor->solicitudes[i + 1];
            }
            receptor->num_solicitudes--;
            return 0;
        }
        else {
            printf( format: "No s'ha pogut acceptar la sol·licitud. La llista d'amics està plena.\\n");
            return -1;
        }
    }
}

```

Per altra banda, si denegam la sol·licitud d'amistat, s'esborra la sol·licitud d'amistat de la llista de sol·licituds de l'usuari i apareix un missatge per consola informant de que s'ha eliminat.

```

else if (opcion2 == 2) {
    printf( format: "Sol·licitud denegada. No ets amic de %.\\n", solicitant -> nom);

    //També hem de fer que esborri la sol·licitud denegada de la llista de sol·licituds actual
    for (int i = opcion - 1; i < receptor -> num_solicitudes - 1; i++) {
        receptor -> solicitudes[i] = receptor -> solicitudes[i + 1];
    }
    receptor -> num_solicitudes--;
    return 0;
}

```

Llistar_amics_acceptats:

Aquesta funció és una funció molt simple que pren com a paràmetres l'struct de l'usuari que té la sessió iniciada.

La funció simplement entra a dins un bucle for que es repeteix tantes vegades com amics hi ha a la llista d'amics de l'usuari i imprimeix per consola els noms dels usuaris presents a la llista. Si la llista està buida, no entra al bucle for i surt de la funció.

```
void llistar_amics_acceptats(User* usuario) { //Funció perquè aparegui en pantalla la llista d'amics acceptats que tens
    printf( format: "Amics acceptats de %s:\n", usuario -> nom);

    if (usuario -> num_amics == 0) {
        printf( format: "No tens amics acceptats.\n");
        return;
    }
    //Fem un bucle perquè vagi llistant als amics
    for (int i = 0; i < usuario -> num_amics; i++) {
        printf( format: "%d. %s\n", i + 1, usuario -> amics[i] -> nom);
    }
    sleep(1.5);
}
```

USUARI.c:

A continuació ens explaiarem de quines han sigut les funcions dirigides als usuaris, aquestes estan dividides en tres subapartats: funcions elementals, publicació i diccionari.

En el cas de les funcions elementals, són aquelles funcions fonamentals que serveixen per ser la base del funcionament de l'usuari dins la nostra xarxa social.

Primer, les funcions dirigides a emmagatzemar les dades dels usuaris als fitxers, són necessàries perquè, gràcies a elles, les dades dels usuaris (tant el perfil personal de l'usuari com la seva llista d'amics i sol·licituds d'amistat) no es perden una vegada es tanca el codi.

Guardar_usuaris_en_arxiu:

Per començar, la funció guardar_usuaris_en_arxiu serveix per copiar les dades dels usuaris a als fitxers pertinents per tal d'evitar que es perdin una vegada es tanca el codi. Aquesta funció és la darrera en executar-se, ja que tan sols s'executa quan es tria l'opció per sortir del codi.

La funció pren com a paràmetre el punter a la llista de tots els usuaris registrats a l'aplicació. D'aquesta forma, es pot iterar a través de tota la llista i obtenir les dades de tots els usuaris registrats. A més, cal remarcar que és necessària l'existència de tres fitxers on emmagatzemar les dades: PERFIL.txt, AMICS.txt i SOL·LICITUDS_AMICS.txt.

El primer que fa la funció, és obrir el fitxer PERFIL.txt. Si no es troba el fitxer, s'imprimeix un missatge d'error per consola i es surt de la funció.

```
if (arxiu == NULL) {
    printf( format: "No s'ha pogut obrir el fitxer.\n");
    return;
}
```

Si el fitxer s'obri correctament, es defineix la variable current: un punter de tipus user que apunta al primer element de la llista d'usuaris. A continuació, s'inicialitza un bucle while que es repeteix mentre la variable current no sigui NULL. A cada iteració del bucle, mitjançant un fprintf, s'escriuen les següents dades del usuari a una sola línia del fitxer: nom, contrasenya, primer i segon cognom, edat, correu electrònic, ubicació, els cinc esports preferits de l'usuari, el número de sol·licituds d'amistat i amics i la quantitat de quilòmetres correguts per l'usuari. A més, abans de finalitzar el bucle, la variable current prèviament definida, es defineix com el següent usuari a la llista d'usuaris existents. D'aquesta forma, el bucle es repeteix fins que ja no queden usuaris a la llista i s'escriuen al fitxer les dades de cada usuari a cada línia. Finalment, es tanca l'arxiu.

```
User* current = Llista -> head;
while (current != NULL) {
    fprintf( stream: arxiu, format: "%s %s %s %s %d
    current = current -> next;
}

fclose( File: arxiu);
```

Una vegada la funció ha acabat el fitxer PERFIL, s'obri el fitxer SOL·LICITUDS_AMICS. En aquest fitxer s'emmagatzemen els noms dels usuaris presents a la llista de sol·licituds d'amistat de l'usuari. Aquesta part segueix un funcionament semblant al de la part anterior. Es torna a definir la variable current com al primer element de la llista d'usuaris registrats i, seguidament s'entra a dins el bucle while, el qual es repetirà tantes vegades com usuaris hi ha a la llista. A dintre del bucle while, hi ha un bucle for anidat que serveix per iterar per tot l'array de sol·licituds d'amistat de l'usuari (current). Per tant, una vegada s'entra al bucle while, s'imprimeix al document el nom de l'usuari (current) i després s'itera pel bucle for imprimint el nom de tots els usuaris a l'array de sol·licituds fins que ja no hi ha més sol·licituds i redefineix current com el següent usuari a la llista d'usuaris registrats. Finalment, es tanca l'arxiu.

```

current = Llista->head;
while (current != NULL) {
    fprintf( stream: arxiu_solicitudes, format: "%s ", current->nom);
    for (int i = 0; i < current->num_solicitudes; i++) {
        fprintf( stream: arxiu_solicitudes, format: "%s ", current->solicitudes[i]->nom);
    }
    fprintf( stream: arxiu_solicitudes, format: "\n");
    current = current->next;
}
fclose( File: arxiu_solicitudes);
  
```

Finalment, s'obri el fitxer AMICS per guardar el nom dels amics de cada usuari a un fitxer. Aquesta part segueix el mateix procediment que la part de les sol·licituds però mitjançant el fitxer AMICS.txt i guardant la variable current->amics[i]->nom de cada usuari en lloc de la variable current->sol·licituds[i]->nom.

```

current = Llista->head;
while (current != NULL) {
    fprintf( stream: arxiu_amics, format: "%s ", current->nom);
    for (int i = 0; i < current->num_amics; i++) {
        fprintf( stream: arxiu_amics, format: "%s ", current->amics[i]->nom);
    }
    fprintf( stream: arxiu_amics, format: "\n");
    current = current->next;
}
fclose( File: arxiu_amics);
  
```

Sabem, que la funció es podria haver implementat usant un sol bucle while per guardar tota la informació als tres fitxer, però varem decidir escriure d'aquesta forma la funció per poder diferenciar millor a quin arxiu es guarda la informació i perquè la part dels fitxers SOL·LICITUDS_AMICS i AMICS va ser implementada després de la part del fitxer PERFIL.

Llegir_usuaris_desde_arxiu:

Aquesta funció serveix per llegir la informació introduïda als fitxers per la funció prèviament definida. Mitjançant aquesta funció, es pot tornar a inicialitzar la llista d'usuaris amb la informació dels arxius una vegada s'ha tancat el programa i la informació s'ha perdut. Aquesta funció, igual que l'anterior, pren com a paràmetre el punter a la llista de tots els usuaris registrats a l'aplicació, encara que en aquest cas la llista està buida i serà la funció la que l'ompleni amb les dades de cada usuari.

La funció comença obrint els arxius PERFIL.txt, SOL·LICITUDS_AMICS.txt i AMICS.txt. Si no es troben els fitxer, s'imprimeix un missatge d'error per consola i es sort de la funció. Si s'obrin bé els fitxers, mitjançant un bucle while amb un fscanf, es llegeixen totes les variables de l'usuari presents al fitxer PERFIL i es copien (strcpy) a l'struct del primer usuari de la llista. Posteriorment, es passa al següent usuari (current = current->next) i es torna a repetir el bucle per emmagatzemar les dades del nou usuari. Això es repeteix tantes vegades com línies té el fitxer PERFIL, ja que cada línia conté la informació d'un usuari.

```

while (fscanf( stream: arxiu,  format: "%s %s %s %s %d %s "
User* user = (User*)malloc( Size: sizeof(User));
strcpy( Dest: user -> nom,  Source: nom);
strcpy( Dest: user -> password,  Source: password);
strcpy( Dest: user -> cognom1,  Source: cognom1);
strcpy( Dest: user -> cognom2,  Source: cognom2);
user -> edat = edat;
strcpy( Dest: user -> correu,  Source: correu);
strcpy( Dest: user -> ubi,  Source: ubi);
strcpy( Dest: user -> gust1,  Source: gust1);
strcpy( Dest: user -> gust2,  Source: gust2);
strcpy( Dest: user -> gust3,  Source: gust3);
strcpy( Dest: user -> gust4,  Source: gust4);
strcpy( Dest: user -> gust5,  Source: gust5);
user -> next = NULL;
Llista -> num_persones++;
user -> num_solicitudes = num_solicitudes;
user -> num_amics = num_amics;
user -> quilometres = quilometres;
```

Una vegada ja s'ha guardat la informació de cada usuari a la llista, falta omplir els arrays d'amics i sol·licituds de cada usuari. Per aquest motiu, la part de la funció relacionada amb els fitxers SOL·LICITUDS_AMICS i AMICS s'ha d'executar una vegada ja s'han guardat les dades dels usuaris des de l'arxiu.

Comencem amb la part de les sol·licituds d'amistat. Inicialment es defineixen dues variables user: user (en aquesta variable s'emmagatzemaran les dades de les sol·licituds d'amistat) i

iterar_llista (serveix per recórrer la llista d'usuaris). Seguidament, entrem a un bucle for que es repeteix tantes vegades com usuaris a la llista d'usuaris. A cada iteració del bucle for, es llegeix una línia del fitxer SOL·LICITUDS_AMICS. Recordem que, a cada línia, el primer string que apareix és el nom de l'usuari i la resta de noms que apareixen són els noms dels usuaris presents a la llista de sol·licituds. Mitjançant un sistema de tokens, cada línia es divideix en tokens (un string per token) i s'emmagatzemen els tokens a un nou array: tokens_sol·licituds[].

```

char linea_solicitudes[500];
if (fgets( Buf: linea_solicitudes, MaxCount: sizeof(linea_solicitudes), File: arxiu_solicitudes) != NULL) {
    linea_solicitudes[strcspn( Str: linea_solicitudes, Control: "\n")] = '\0';

    // Hem de dividir les línies de l'arxiu en tokens, 1 token per a cada nom (el primer no es té en compte)
    char *tokens_solicitudes[MAX_SOLICITUDS];
    int contadorTokensSolicitudes = 0;

    char *token_solicitudes = strtok( Str: linea_solicitudes, Delim: " " );
    while (token_solicitudes != NULL && contadorTokensSolicitudes < MAX_SOLICITUDS) {
        tokens_solicitudes[contadorTokensSolicitudes] = token_solicitudes;
        contadorTokensSolicitudes++;
        token_solicitudes = strtok( Str: NULL, Delim: " " );
    }
}
  
```

Finalment, mitjançant un altre bucle for amb un for anidat a dins, es recorr la llista d'usuaris (mitjançant la variable iterar_llista) comparant els strings presents a l'array tokens_sol·licituds amb els noms dels usuaris de la llista d'usuaris. Cada vegada que hi ha una coincidència, vol dir que el nom que està al fitxer pertany a aquella persona de la llista d'usuaris i aquesta persona es guarda a l'array de sol·licituds del user.

```

// Es guarden els valors dels tokens a user->solicitudes
int c = 0;
for(int i = 1; i < contadorTokensSolicitudes; i++){
    iterar_llista = Llista->head;
    for(int j = 0; j < Llista->num_persones; j++) {
        if (strcmp(iterar_llista->nom, tokens_solicitudes[i]) == 0) {
            user->solicitudes[c] = iterar_llista;
            c++;
        }
        iterar_llista = iterar_llista->next;
    }
}
user = user->next;
iterar_llista = Llista->head;
}
  
```

Posteriorment, es repeteix aquesta part del codi però amb el fitxer AMICS. Es segueix la mateixa estratègia de llegir el fitxer per línies i guardar els strings com a tokens per després

comparar-los. La única diferència és que en lloc de llegir les dades del fitxer SOL·LICITUDS_AMICS, les llegeix del fitxer AMICS.

```

user = Llista->head;
iterar_llista = Llista->head;
for(int p = 0; p < Llista->num_persones; p++) {

    // Llegir el fitxer AMICS.txt per línies
    char linea_amics[500];
    if (fgets( Buf: linea_amics, MaxCount: sizeof(linea_amics), File: arxiu_amics) != NULL) {
        linea_amics[strcspn( Str: linea_amics, Control: "\n")] = '\0';

        // Dividir les línies del fitxer AMICS.txt en tokens (usant la funció strtok i usant
        char *tokens_amics[MAX_AMICS];
        int contadorTokensAmics = 0;

        char *token_amics = strtok( Str: linea_amics, Delim: " ");
        while (token_amics != NULL && contadorTokensAmics < MAX_SOLICITUDES) {
            tokens_amics[contadorTokensAmics] = token_amics;
            contadorTokensAmics++;
            token_amics = strtok( Str: NULL, Delim: " ");
        }
    }
}
  
```

```

// Guardar els tokens a user->amics
int c = 0;
for(int i = 1; i < contadorTokensAmics; i++){
    iterar_llista = Llista->head;
    for(int j = 0; j < Llista->num_persones; j++) {
        if (strcmp(iterar_llista->nom, tokens_amics[i]) == 0) {
            user->amics[c] = iterar_llista;

            c++;
        }
        iterar_llista = iterar_llista->next;
    }
}
user = user->next;
iterar_llista = Llista->head;
}
  
```

Finalment, es tanquen els tres fitxers i finalitza la funció.

Emmagatzema_dades:

Aquesta funció serveix per guardar les dades dels usuaris que es registren novament a la xarxa social. La funció pren com a paràmetres: el punter a la llista d'usuaris i el punter a una variable user.

Aquesta funció consisteix en un conjunt de parells printf i scanf. Es demana a l'usuari qui és la seva contrasenya i l'usuari escriu per consola la seva contrasenya la qual es guarda a la variable usuari->password.

```
printf( format: "Introdueix la contrasenya: \n");
scanf( format: "%s", usuari -> password);

printf( format: "Introdueix el teu primer cognom: \n");
scanf( format: "%s", usuari -> cognom1);

printf( format: "Introdueix el teu segon cognom: \n");
scanf( format: "%s", usuari -> cognom2);

printf( format: "Introdueix la teva edat: \n");
scanf( format: "%d", &usuari -> edat);

printf( format: "Introdueix el teu correu electrònic: \n");
scanf( format: "%s", usuari -> correu);
```

Algunes d'aquestes variables presenten estratègies per evitar errors. Per exemple, la variable nom es demana mitjançant un bucle while que es repeteix mentre el nom no sigui el mateix que el d'un usuari ja present a la llista d'usuaris o la variable del correu electrònic.

```
while(1) {
    printf( format: "Introdueix el teu nom: \n");
    scanf( format: "%s", usuari -> nom);

    if (comprovar_usuari( llista: Llista, usuari -> nom)) {
        printf( format: "L'usuari ja existeix. \n");
    } else {
        break;
    }
}
```

Finalment, ja que és un usuari nou, les variables relacionades amb la part social (num_amics, num_solicitudes, quilòmetres, etc) s'inicialitzen com a zero.

Print_user_info:

La funció *print_user_info* té com a finalitat imprimir per pantalla tota la informació personal de l'usuari. Es crida la funció amb el paràmetre del punter current a l'estructura d'*User*,

`void print_user_info(User *current).`

Les variables utilitzada és:

- *current*: punter a l'estructura *User* que conté les dades de l'usuari a mostrar

S'imprimeixen totes les dades personal de l'usuari accedint a tots els llocs pertinents de l'usuari *current*.

```
void print_user_info(User *current) {
    printf( format: "\n");
    printf( format: "=====|\n");
    printf( format: "| Dades personals |\n");
    printf( format: "| ======|\n");
    printf( format: "| Nom: %s\n", current->nom);
    printf( format: "| Cognom: %s\n", current->cognom1);
    printf( format: "| 2n Cognom: %s\n", current->cognom2);
    printf( format: "| Edat: %d\n", current->edat);
    printf( format: "| Correu: %s\n", current->correu);
    printf( format: "| Ubicacio: %s\n", current->ubi);
    printf( format: "| ======|\n");
    printf( format: "| Esports |\n");
    printf( format: "| ======|\n");
    printf( format: "| 1. %s\n", current->gust1);
    printf( format: "| 2. %s\n", current->gust2);
    printf( format: "| 3. %s\n", current->gust3);
    printf( format: "| 4. %s\n", current->gust4);
    printf( format: "| 5. %s\n", current->gust5);
    printf( format: "| ======|\n");
    printf( format: "| Kilometres |\n");
    printf( format: "| ======|\n");
    printf( format: "| Km totals: %d\n", current->quilometres);
    printf( format: "| ======|\n");
}
```

La funció està definida entre les línies 264 i 288 (*USUARI.c*) i a la línia 14 (*USUARI.h*). I s'utilitza en la línia 142 del *menu.c* a la funció *menu_usuari*, que una vegada s'ha iniciat sessió amb un usuari i haver elegit l'opció 1 (Perfil) es mostren per pantalla tota la informació i després et demana si vols modificar aquestes mateixes dades amb la funció *canvi_de_dades*.

Canvi_de_dades:

La funció *canvi_de_dades*, com el seu nom indica, permet a l'usuari a modificar les seves dades personals. Es crida amb el punter *current* a l'estructura User `void canvi_de_dades(User *current)`.

Les variables utilitzades són:

- *option_3_1*: array de caràcters que guarda l'opció (submenú) introduïda per l'usuari
- *current*: punter a l'estructura User per accedir a la dada a modificar
- *nova_edat*: variable *int* per guardar la nova edat i verificar que és correcta (un enter)
- *nou_correu*: variable *char* per guardar el nou correu i verificar que és correcta

Primer, demana a l'usuari quina és la dada que vol modificar (que es guarda en *option_3_1*) que es passa a minúscula (amb la funció *tolower*) per evitar errors i comprovar si coincideix amb alguna de les dades (amb *strcmp*). I en el cas de coincidir en alguna, demana que s'introdueixi un nou valor i aquest es guarda on toca. En cas contrari, es mostra un missatge d'error i torna a demanar, fins a proporcionar una opció vàlida, quina és la dada a canviar. En el cas de l'edat i el correu, hem de verificar que les dades introduïdes són vàlides.

La funció està definida entre les línies 290 i 383 (USUARI.c) i a la línia 15 (USUARI.h). I s'utilitza en la línia 145 del menu.c a la funció *menu_usuari*, que una vegada s'ha iniciat sessió amb un usuari i haver elegit l'opció 1 (Perfil) es mostren per pantalla totes les dades personal i demana si les vols modificar (si la funció *resp_bol* és un 1) o no (*resp_bol* = 0).

```
void canvi_de_dades(User *current) {
    char option_3_1[MAX_LENGTH];
    printf( format: "Quina? ");
    while (1) {
        scanf( format: "%s", option_3_1);

        // Convertim l'opció introduïda a minúscula
        for (int i = 0; option_3_1[i]; i++) {
            option_3_1[i] = tolower( C option_3_1[i]);
        }

        if (strcmp(option_3_1, "nom") == 0) {
            printf( format: "Ja pots introduir el canvi: ");
            scanf( format: "%s", current -> nom);
            break;
        } else if (strcmp(option_3_1, "cognom") == 0) {
            printf( format: "Ja pots introduir el canvi: ");
            scanf( format: "%s", current -> cognom);
            break;
        } else if (strcmp(option_3_1, "2cognom") == 0 || strcmp(option_3_1, "2ncognom") == 0) {
            printf( format: "Ja pots introduir el canvi: ");
            scanf( format: "%s", current -> cognom2);
            break;
        }
    }
}
```

```
    } else if (strcmp(option_3_1, "edat") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        int nova_edat;

        while (1) { // comprovar que introduïm un enter per a la edat
            if (scanf( format: "%d", &nova_edat) != 1 || nova_edat < 0 || nova_edat > 120) {
                printf( format: "Valor d'edat no valid. Introduceix un enter: \n");
                while (getchar() != '\n');

            } else {
                current -> edat = nova_edat;
                break;
            }
        }
        break;

    } else if (strcmp(option_3_1, "correu") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        char nou_correu;
        scanf( format: "%s", &nou_correu);

        while (1) {
            if (comprovar_correu( correu: &nou_correu) == true) { // funció per validar que el correu està ben escrit
                printf( format: "El correu es valid.\n");
                strcpy( Dest: current -> correu, [Source: &nou_correu]); // copiem el que hem escrit a les dades de l'usuari
                break;
            } else {
                printf( format: "El correu es invalid. Torna a introduir el teu correu: \n");
                scanf( format: "%s", &nou_correu);
            }
        }
        break;

    } else if (strcmp(option_3_1, "ubicacio") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        scanf( format: "%s", current -> ubi);
        break;

    } else if (strcmp(option_3_1, "gust1") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        scanf( format: "%s", current -> gust1);
        break;

    } else if (strcmp(option_3_1, "gust2") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        scanf( format: "%s", current -> gust2);
        break;

    } else if (strcmp(option_3_1, "gust3") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        scanf( format: "%s", current -> gust3);
        break;

    } else if (strcmp(option_3_1, "gust4") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        scanf( format: "%s", current -> gust4);
        break;

    } else if (strcmp(option_3_1, "gust5") == 0) {
        printf( format: "Ja pots introduir el canvi: ");
        scanf( format: "%s", current -> gust5);
        break;

    } else {
        printf( format: "Opció incorrecta, introduceix una valida: ");
    }
}
```

A continuació, explicarem les funcions dedicades a l'organització de publicacions així com la gestió de les mateixes.

fer_publicacio:

Aquesta funció està dissenyada per tal de permetre a l'usuari realitzar publicacions, el que en el marc intern del codi consisteix a crear una pila que emmagatzema cadenes de caràcters per tal d'emmagatzemar totes les publicacions de l'usuari.

Els paràmetres d'entrada emparades són un punter de memòria a una estructura d'usuari i un punter de memòria a una estructura de diccionari. En primer lloc usem el punter de memòria a l'estructura d'usuari per tal de guardar la publicació en la llista de publicacions de l'usuari en qüestió. D'altra banda, el punter al diccionari ens serveix per anar emmagatzemant totes les paraules que conformen cada publicació de l'usuari.

En prime lloc demanem a l'usuari que introdueixi en la taula de comandes la publicació que vol realitzar, tot seguit emmagatzem el que l'usuari escriu en una cadena de caràcters. En segon lloc, fem una crida a la funció “quilometres” la qual preguntarà i emmagatzemarà quants quilòmetres ha recorregut l'usuari.

Més endavant, mitjançat memòria dinàmica, afegirem aquesta publicació a la llista dinàmica de publicacions de l'usuari. Finalment, mitjançant un bucle que ens permet analitzar cada paraula de la publicació, es mirarà mitjançat la crida a la funció “buscar_paraula” si aquella paraula ja rau en la estructura la qual apunta “Taula”. En cas afirmatiu, s'implement s'iterarà en contador d'aquella paraula en l'estructura que apunta “Taula”. En cas contrari, s'afegira aquesta nova paraula al diccionari mitjançant la funció “afegir_paraula_nova”.

```
void fer_publicacio(User* usuari, st_Diccionari* Taula) { //Emmagatzemem la publicacio
    char text[MAX_CHARACTERS + 1];
    printf("Introdueix el text de la publicacio (maxim %d caracters)", MAX_CHARACTERS);
    scanf("%[^\\n]", text);
    quilometres(usuari);

    Publicacio* nova_publicacio = (Publicacio*) malloc(sizeof(Publicacio)); //Creem una nova publicacio
    strcpy(Dest: nova_publicacio -> text, Source: text); //Insertem el text en la nova publicacio
    nova_publicacio -> seguent = usuari -> publicacio.top;
    usuari -> publicacio.top = nova_publicacio;

    //Dividim la publicacio en paraules i inserim cada una d'elles en el diccionari.
    char copia_text[MAX_CHARACTERS + 1];
    strcpy(Dest: copia_text, Source: text);
    char* token = strtok(Str: copia_text, Delim: " ");
    while (token != NULL) {
        Paraula* existent = buscar_paraula(Taula, word: token); //Es busca la paraula existent
        if (existent == NULL) { //En cas de no trobar la paraula
            if (Taula -> num_paraules < MAX_PARAULES) { //Si no s'ha arribat al límit
                afegir_paraula_nova(Taula, word: token); //Afegim la paraula nova
            }
        }
        token = strtok(Str: NULL, Delim: " ");
    }
}
```

Timeline:

La funció Timeline és una funció que careix de complexitat. La seva finalitat és imprimir en la consola totes les publicacions realitzades per l'usuari.

La funció consta únicament d'un paràmetre d'entrada, un punter de memòria a una estructura "User". D'aquesta manera s'accedeix a la seva llista de publicacions i amb un bucle while, iterem mentre anem imprimint cada cadena de caràcters emmagatzemada.

Per poder realitzar aquesta funció en primer lloc ens hem d'assegurar començar imprimint per la primera publicació de tota la llista, això es fa mitjançant la següent línia:

```
Publicacio* publicacio_actual = usuari -> publicacio.top;
```

```
void Timeline (User* usuari) {
    printf( format: "Timeline de %s:\n", usuari -> nom);

    Publicacio* publicacio_actual = usuari -> publicacio.top;

    while (publicacio_actual != NULL) {
        printf( format: "- %s\n", publicacio_actual -> text);
        publicacio_actual = publicacio_actual -> seguent;
    }

    printf( format: "Fi del Timeline.\n");
}
```

quilometres:

La funció *quilometres* registra els quilòmetres que ha recorregut un usuari. Es crida amb el punter a l'estructura *User*, `void quilometres(User* usuari)`.

La variable que utilitza és, *quilometres*, enter per guardar els nombre de quilòmetres ha recorregut l'usuari aquell dia.

Primer demana quants de quilòmetres hem fet i entra al bucle infinit per respondre correctament (escriure un enter per consola, no major a 500). En cas de que s'ha introduït una resposta vàlida, guarda el valor dins les dades personals de l'usuari (en sumatori).

La funció està definida entre les línies 427 i 441 (*USUARI.c*) i a la línia 18 (*USUARI.h*). I s'utilitza en la línia 391 del *USUARI.c* a la funció *fer_publicacio*, després d'haver escrit la publicació, s'inicia la funció *quilometres* per registrar els quilòmetres recorreguts.

```
void quilometres(User* usuari) {
    int quilometres;
    printf(format: "Quants quilometres has recorregut avui? ");

    while (1) { // comprovar que introduim un enter per als quilometres
        if (scanf(format: "%d", &quilometres) != 1 || quilometres < 0 || quilometres > 500) {
            printf(format: "Valor de quilometres no valid. Introduceix un enter: \n");
            while (getchar() != '\n');

        } else {
            usuari -> quilometres += quilometres;
            break;
        }
    }
}
```

print_rankingKM:

La funció *print_rankingKM*, imprimeix per pantalla una llista (en forma de rànquing) dels usuaris que han fet més quilòmetres. `void print_rankingKM(user_list* Llista, User* usuari)`.

Les variables utilitzades són:

- *Llista*: punter a l'estructura de dades que conté la llista d'usuaris
- *usuari*: punter a l'usuari actual
- *usuaris*: array de punters a usuaria per guardar els noms pel rànquing
- *numUsuaris*: nombre enter d'usuaris en la llista

Primer es crea una array d'*usuaris* i s'inicialitza la variable *numUsuaris* a 0. Per després anar recorrent la llista i comptar quants d'usuaris hi ha i guardar el nombre a *numUsuaris*.

S'ordenen els usuaris de la llista de més quilòmetres recorreguts a menys i s'imprimeix el rànquing.

La funció està definida entre les línies 443 i 467 (USUARI.c) i a la línia 19 (USUARI.h). I s'utilitza en la línia 179 del menu.c a la funció *menu_usuari*, si s'ha elegit la opció 8. Ranking km recorreguts, s'inicia la funció i imprimeix el rànquing.

```
void print_rankingKM(user_list* Llista , User* usuari) {
    User* usuaris[MAX_USERS];
    int numUsuaris = 0;

    User* current = Llista -> head;
    while (current != NULL && numUsuaris < MAX_USERS) {
        usuaris[numUsuaris] = current;
        current = current -> next;
        numUsuaris++;
    }

    for (int i = 0; i < numUsuaris - 1; i++) {
        for (int j = 0; j < numUsuaris - i - 1; j++) {
            if (usuaris[j] -> quilometres < usuaris[j + 1] -> quilometres) {
                User* temp = usuaris[j];
                usuaris[j] = usuaris[j + 1];
                usuaris[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < numUsuaris; i++){
        printf( format: "%d. %s amb %d quilometres\n", i + 1, usuaris[i] -> nom, usuaris[i] -> quilometres);
    }
}
```

Finalment, tenim les funcions dedicades al diccionari:

En primer lloc, ens topem amb la funció “swap”. Aquesta funció declarà dos variables, la variable ”x” la qual és un punter a una struct Paraula. La funció pren per finalitat intercanviar dops adreces de memòria apuntades per ”a” i ”b”, que són punters a punters de ”Paraula”. Afirmem que aquesta funció especifica no pateix ninguna limitació significativa ni requereix de millores degut a la simpleza del seu codi.

```
void swap(Paraula** a, Paraula** b) {
    Paraula* x = *a;
    *a = *b;
    *b = x;
}
```

En segon lloc, la funció ”partició” declara dues variables. La primera és un enter declarat com ”i” i la segona un punter a una estructura ”Paraula”, aquesta última la declarem com ”pivot”. La funció realitza una partició en l’array de punters de ”Paraula” entre els índex ”bot” i ”top” i retorna un índex de partició. La principal limitació és que la funció assumeix que l’array ”a” conté adreces de memòria vàlides i no detecta límits específics. Podriem millorar-lo afegint una validació per assegurar-be que ”bot” i ”top” estan dins dels límits de l’array ”a”.

```
int particio (Paraula** a, int bot, int top)
int i = bot - 1;
Paraula* pivot = a[top];

for(int j=bot; j<top; j++) {
    if (a[j] -> cont > pivot -> cont) {
        i++;
        swap( a: &a[i], b: &a[j]);
    }
}
swap( a: &a[i+1], b: &a[top]);
return i+1;
}
```

Com podem veure, per al seu funcionament necessitem tres paràmetres d’entrada. El primer serà un punter a un punter a una estructura de dades ”Paraula”. El segon i tercer seran dos paràmetres enters que ens ajudaran com a marges del nostre diccionari.

A continuació, la funció quicksort, te per finalitat ser la funció la qual ordeni de manera recursiva la pila de paraules per tal d’implementar la funció que imprimirà les paraules TOP. Aquesta funció crida la funció partició per tal de trobar l’element pivot. Al igual que en la funció ”partició” podriem haver afegit una validació per assegurar-nos que ”bot” i ”top” estan dins dels límits de l’array ”a”.

```
void quicksort (Paraula** a, int bot, int top) {
    if (bot < top){
        int pivot = particio(a, bot, top);
        quicksort(a, bot, top: pivot - 1);
        quicksort(a, bot: pivot + 1, top);
    }
}
```

Una de les funcions més importants respecte les mecàniques de les publicacions serà la funció “trending” la qual simplement rebut un diccionari amb totes les paraules ordenades de més usades a menys usades pels usuaris, les imprimirà una darrera l’altra. Això serà possible gràcies a la crida de la funció quicksort. Podriem afegir una comprovació per assegurar-nos que el paràmetre “st_Diccionari” no és NULL.

```
void trending (st_Diccionari* diccionari) {
    if (diccionari -> num_paraules == 0){
        printf( format: "Encara no s'ha realitzat ninguna publicacio");
        return;
    }
    quicksort( a: diccionari -> paraules, bot: 0, top: diccionari -> num_paraules - 1);

    printf( format: "\nTop paraules:\n");
    for(int i = 0; i < diccionari -> num_paraules && i < 10; i++) {
        printf( format: "La paraula '%s', s'ha utilitzat %d.\n", diccionari -> paraules[i] -> paraula, diccionari -> paraules[i] -> cont);
    }
}
```

Tot seguit, un cop l’usuari ha fet una publicació es crida la funció “buscar_paraula” la qual buscarà en tot el diccionari cada paraula de la publicació realitzada. La seva finalitat es saber si s’haurà de sumar un nou element al diccionari o s’hi s’ha de sumar el comptador de d’alguna paraula ja enregistrada en el diccionari. En aquesta funció declarem la variable “i” com a enter, la qual ens farà d’índex per anar iterant en el diccionari.

```
Paraula* buscar_paraula (st_Diccionari* Taula, char* word) {
    for (int i = 0; i < Taula -> num_paraules; i++) {
        if (strcmp(Taula -> paraules[i] -> paraula, word) == 0) {
            Taula -> paraules[i] -> cont++;
            return Taula -> paraules[i];
        }
    }
    return NULL;
}
```

Finalment, la funció “afegir_paraula_nova” és una funció que serà cridada en cas que la paraula que s'estigui tractant sigui una no existent dintre el diccionari. En aquest cas, es crearà una nova estructura “Paraula” per tal de guardar-hi aquesta nova paraula trobada i es definirà el recompte d'aquella paraula a 1. Podriem millorar aquesta funció comprovant que el paràmetre “Taula” no és NULL abans de processar-lo.

```
void afegir_paraula_nova (st_Diccionari* Taula, char* word) {           //Afegeix una paraula nova
    Paraula* new_paraula = (Paraula*) malloc( sizeof(Paraula));           //Declarem una variable
    strcpy( Dest: new_paraula -> paraula, Source: word);                  //Copiem la paraula
    new_paraula -> cont = 1;                                              //Declarem el recompte
    Taula -> paraules[Taula -> num_paraules] = new_paraula;             //Copiem en la taula
    strcpy( Dest: Taula -> paraules[Taula -> num_paraules] -> paraula, Source: word); //Copiem en la taula
    Taula -> num_paraules++;                                            //Sumem el recompte
}
```

USUARI.h:

En el fitxer USUARI.h es declaren les funcions definides en el fitxer USUARI.c

REFERÈNCIES

Per fer possible el codi de la xarxa social, hem utilitzat pàgines web com w3Schools o geeksforgeeks per cercar informació de com aplicar algunes funcionalitats. I també hem fet ús del Chat GPT per al gestionament d'errors i possibles millors a fer en les funcions.

- "OpenAI. (2023). ChatGPT. [Software de inteligencia artificial]. Recuperado de <https://openai.com/>"
- GeeksforGeeks. (s. f.). GeeksforGeeks | A computer science portal for geeks. <https://www.geeksforgeeks.org/>
- W3Schools Online Web Tutorials. (s. f.). <https://www.w3schools.com/>