Universitat Pompeu Fabra Barcelona

ESTRUCTURAS DE DADES I ALGORITMES II, 2022-2023

La nostra xarxa social

Nombre de los estudiantes:

Gerard Borràs Portell

Joan Josep Díaz Vega

Número de los estudiantes:

U213929

U214375

Fecha de entrega:

07/06/2023

LLISTA DE CONTINGUTS

INTRODUCCIÓ

Un cop ja havent-hi fet Estructura de Dades i Algorismes I (EDA I), els estudiants ja estem acostumats a fer petites funcions independents que resolguessin problemes puntuals. Però encara s'havia de veure si podiem fer un programa sencer amb les seves pròpies mans. Per això l'objectiu de les pràctiques d'Estructura de Dades i Algorismes II(EDA II) era fer un prototip de xarxa social.

Aquest prototip havia de contenir un menú on es pugui registrar un usuari mitjançant un fitxer o manualment un a un, accedir a una llista de tots els usuaris i accedir a un menú d'usuari. Per entrar aquest menú farà introduïr un identificador. Aquest menú personal haurà de contenir obligatòriament una opció per enviar sol·licituds d'amistat, una per aceptarles, una per publicar un post, una altra per llistar les publicacions d'un usuari en concret i una opció per tornar al menú principal.

Abans de començar el projecte els diferents integrants del grup varem fer una pluja d'idees la qual no va portar a ningun lloc ja que no sabíem les limitacions que tindríem al programar la pagina web o sigui que varem decidir primer fer la pàgina web i després fer la pluja d'idees i maquillar la pagina web per donar-li una identitat unica. Així doncs varem començar el programa fent estrictament el que ens deia l'enunciat.

OBJECTIUS DEL PROJECTE

Per començar vàrem crear el projecte en CLion i el vam vincular amb Github. Però la vinculació amb Github ens va resultar molt complicada i vam decidir deixar-ho per a després (no vam veure el pdf on ens explica de manera senzilla com vincular-lo). Així doncs, vàrem començar amb la funció main on es cridava a la funció *menu()*;. També ens vam assegurar

d'importar totes les llibreries i definir alguns conceptes com TRUE, FALSE i MAX_LENGTH(3-12) que aquest últim tindria un valor de 20 i s'encarregava de limitar el nombre de caràcters en un string

```
int main(){
    printf( format: "\nBienvenido a nuestra aplicacion\n");
    Admininicialization();
    menu();
}
```

Funció main (La funció Admininicialization es va afegir més endavant)

Tot seguit en la funció *menu()*(1017-1066) vàrem fer diversos prints que mostressin en pantalla les opcions en forma de menú i vàrem fer un scanf i diversos ifs que detectessin quina opció seleccionessin. Un cop fet el menú teníem clar que la primera funció que havíem de fer en el nostre menú, ignorem la funció de sortir del menú, seria la d'inicialitzar un usuari.

Per fer aquesta funció abans necessitaríem fer una estructura que emmagatzema la informació de l'usuari i una manera de passar per tots ells. Així doncs, vàrem fer dues estructures de dades que formessin una llista, la primera estructura (*Node*)(15-30) era l'encarregada d'emmagatzemar les dades de l'usuari i quin era l'usuari anterior i el següent. La següent estructura (*Arraynode*)(34-37) contenia el primer i l'últim Node de la llista per així tenir un ordre. Abans d'obrir el menú creem una variable global de tipus *arrayNode* anomenada usuarios que comenci sent nul·la, per així en un futur anar emmagatzemant els usuaris allà.

```
//Nodo es la estructura inicial. Esta contiene los datos del usuario correspondiente.
struct Node {
    char nombre[MAX_LENGTH];
    //EXTRA: hemos hecho puedas ponter una contraseña en tu perfil y asi solo poder entrar con esta.
    char password[MAX_LENGTH];
    bool ispass;
    int num;
    int edad;
    char mail[MAX_MAIL_LENGHT];
    char ciudad[MAX_LENGTH];
    char musica[4][MAX_LENGTH];
    struct Node* siguiente;
    struct Node* previo;
    struct RequestQueue* solicitudes;
    struct FriendsQueue* Amigos;
    struct Post* lista;
};


// Una estructura que almacena el primer y último usuario guardado. Esta estructura se utiliza para recorrer todos los usuarios.
struct Arraynode{
    struct Node* first;
    struct Node* last;
```

Struct Node i Arraynode encargades de ordenar i definir la informació dels usuaris

Un cop definides les estructures ja podem fer la funció que vagi creant usuaris. Aquesta funció és senzilla, ja que simplement és preguntar a l'usuari les diferents dades que es necessiten per crear el perfil(560-638) i assignar aquestes a una variable de tipus Node(524-556). A l'hora de crear aquestes variables s'ha de tenir en compte que per cada nou usuari que es vagi creant la variable *usuarios* haurà de canviar el valor del seu *Node last* i el que abans era el *Node last* ha de tenir el nou *last* com a *siguiente*.

```
void newuser(char* User,int edad, char* mail, char* ciudad, char musica[][MAX_LENGTH]){
    Node* newuser = new Node;
    strncpy( Dest: newuser->nombre, Source: User, Count: MAX_LENGTH);
    newuser->edad = edad;
    strncpy( Dest: newuser->mail, Source: mail, Count: MAX_LENGTH);
    strncpy( Dest: newuser->ciudad, Source: ciudad, Count: MAX_LENGTH);
    newuser->Amigos =(struct FriendsQueue*) malloc( Size: sizeof(struct FriendsQueue));
    newuser->Amigos->primero = NULL;
    newuser->Amigos->ultimo = NULL;
    newuser->solicitudes =(struct RequestQueue*) malloc( Size: sizeof(struct RequestQueue));
    newuser->solicitudes->primero = NULL;
    newuser->solicitudes->ultimo = NULL;
    newuser->ispass=FALSE;
    memset( Dst: newuser->password, Val: 0, Size: sizeof( newuser->password));

    for(int i=0; i<4; i++ ){
        strncpy( Dest: newuser->musica[i], Source: musica[i], Count: MAX_LENGTH);
    }

    newuser->siguiente = NULL;
    newuser->previo = usuarios.last;

    if(usuarios.last){
        usuarios.last->siguiente = newuser;
        usuarios.last = newuser;
        newuser->num = newuser->previo->num + 1;

    } else{
        usuarios.first = newuser;
        usuarios.last = newuser;
        newuser->num = 1;
    }
```

Funció newuser que crea una variable de tipus Node

Un cop ja tenim la manera d'inicialitzar els usuaris no és gaire difícil fer les altres opcions. Per imprimir la llista d'usuaris simplement hem d'imprimir el nom del primer usuari de l'array usuarios i després imprimir el que va després. I així consecutivament fins que veiem que l'usuari en el qual estem sigui l'últim(718-729).

```
void listuser() {
    char option;
    Node* user = usuarios.first;
    printf(format: "Lista usuarios:\n");
    while (user) {
        printf(format: "ID %d: %s\n", user->num, user->nombre);
        user= user->siguiente;
    }
    printf(format: "Introduzca cualquier caracter para continuar:");
    scanf(format: "%s",&option);
    return;
}
```

Funció listuser

Pel cas d'obrir el menú d'usuari la cosa es complica una mica, ja que per accedir a aquest havíem de fer un algorisme de recerca per això per a cada usuari li vam crear una variable *id* dins de l'estructura *Node* que s'emmagatzema de manera que el primer usuari registrat tingui la *id* 1 el segon la *id* 2 i així progressivament. Sabem perfectament que l'algorisme de recerca es podria haver fet amb el nom, però consideràvem que això absorbiria molts recursos, ja que no és un algorisme que no faríem servir una vegada. Així doncs, un cop tenint la variable *id* l'algorisme de recerca es simplifica simplement a què a partir del primer *Node* d'usuarios moure'ns n-1 vegades el valor de la id introduïda.(385-394)

```
Node *operuserfound(int id){
    Node* actualuser;
    actualuser=usuarios.first;
    if(id>usuarios.last->num) return NULL;

    for(int i=1;i<=id-1;i++){
        actualuser=actualuser->siguiente;
    }
    return actualuser;
}
```

Funció operuserfound

Abans de fer el menú d'usuari vàrem fer una funció que et permetia crear usuaris a partir d'un fitxer. D'aquesta funció el més difícil va ser obrir el fitxer, ja que això ens va portar molts

problemes. Per solucionar aquest problema vàrem haver de revisar un dels seminaris fets en EDA I, un cop el fitxer ja s'obria correctament la part de llegir el fitxer va ser fàcil. Vam agafar com a caràcter de separació el ";" per així poder separar el nom, l'edat, etc. . Un cop ja tenim les variables separades cridem a la funció newuser() (mostrada anteriorment) per crear el nou usuari i si es detecta un altre ; es torna a fer fins que el fitxer es queda NULL.(647-716)

```
void initfuser() {
    char nombreArchivo[100], nombre[MAX_LENGTH], correo[MAX_MAIL_LENGHT], hobby[4][MAX_LENGTH], edadstr[MAX_LENGTH];
    char caracter, ciudad[MAX_LENGTH], separation = ';';
    int edad, i = 0, j = 0;
    FILE *archivo;

    printf( format: "Ingresa el nombre del archivo: ");
    scanf( format: "%s", nombreArchivo);
    archivo = fopen( Filename: nombreArchivo, Mode: "r");

    if (archivo == NULL){
        printf( format: "No se pudo abrir el archivo.\n");
        return;
    }
    printf( format: "Contenido del archivo: \n");
    fgetc( File: archivo);
    fgetc( File: archivo);
    fgetc( File: archivo);
    while ((caracter = fgetc( File: archivo)) != EOF){
```

```
            i = 0;
            j++;
            if (j>7){
                newuser( User: nombre,edad, mail: correo,ciudad, musica: hobby);
                j=0;
            }
        } else{
            switch (j) {
                case 0:
                    nombre[i]= caracter;
                    break;
                case 1:
                    edadstr[i] = caracter;
                    break;
                case 2:

                    correo[i] = caracter;
                    break;
                case 3:
                    ciudad[i] = caracter;
                    break;
                default:
                    hobby[j - 4][i] = caracter;
                    break;

            }
            i++;
        }
    }
    fclose( File: archivo);
```

Funció initfuser()

Un cop feta la funció initfuser(), vam començar a fer el menú d'usuari el qual hauria de tenir l'opció d'enviar sol·licitud d'amistat, l'opció d'acceptar i refusar les sol·licituds pendents, publicar un post, veure els posts d'una persona i tornar al menú principal.(815-950)

En el nostre cas vàrem fer primer la part d'enviar i acceptar sol·licituds d'amistats. Per això igual que a l'hora de crear la funció newuser(); vàrem crear primer unes estructures però en aquest cas tant les sol·licituds d'amistats com les amistats seguirien l'estructura d'una cua.

En el cas de les estructures dels amics tenen una estructura (Friend)(45-48) que conté el Node amic i el següent Friend. I per ordenar els diferents Friends estructura tenim una segona estructura (FriendQueue)(51-54) on s'emmagatzema el primer Friend i l'últim.

```
struct Friend{
    Node* amigo;
    Friend* next;
};

struct FriendsQueue{
    Friend* primero;
    Friend* ultimo;
};
```

En el cas de les sol·licituds d'amistats seria bastant similar, estructura tipo cua que en el primer cas (Request)(89-93) té un Node sol·licitant un Node receptor i el següent Request. I després una segona estructura (requestQueue)(96-99) que emmagatzema el primer Request i l'últim

```
struct Request{
    struct Node* solicitante;
    struct Node* receptor;
    struct Request* siguiente;
};

struct RequestQueue{
    Request* primero;
    Request* ultimo;
};
```

Un cop tenim les estructures fetes vam fer primer la part d'enviar les sol·licituds preguntem a l'usuari a qui li vol enviar la sol·licitud i mitjançant la funció operuserfound ja definida anteriorment el busquem i de manera similar a quant creàvem un nou usuari es crea un nou request amb sol·licitant user i receptor =operuserfound(id)(848-862) la següent request s'estableix com a nul·la. I en el cas que el primer Request del RequestQueue sigui nul aquest passa a ser la nova Request que estem fent i en tots els casos s'estableix el següent

Request de l'últim request com al nou Request i aquest nou últim request passa a ser l'últim.(169-188)(127-140)

```
// PARA ENCOLAR UNA NUEVA SOLICITUD DE AMISTAD
void enqueueRequest(struct RequestQueue* queue, struct Node* solicitante, struct Node* receptor){
    struct Request* nuevaSolicitud = (struct Request*) malloc( Size: sizeof(struct Request));
    nuevaSolicitud->solicitante = solicitante;
    nuevaSolicitud->receptor = receptor;
     nuevaSolicitud->siguiente = NULL;

    if (queue->primero == NULL){
        queue->primero = nuevaSolicitud;
        queue->ultimo = nuevaSolicitud;
    } else {
        queue->ultimo->siguiente = nuevaSolicitud;
        queue->ultimo = nuevaSolicitud;
    }
}
```

Funció enqueueRequest

Un cop fet això fem la part de gestionar les sol·licituds d'amistat. Aquesta part et portarà a un altre submenú on es mostra la primera *Request* que té l'usuari(si té) i es mostren quatre diferents opcions per elegir, aceptar sol·licitud , refusar-la, veure la següent sol·licitud o tornar al menú d'usuari. Si esculls la tercera simplement canviem la variable *Request* per la seva següent.(400-444)

```
while (solicitud != NULL){
    struct Node* solicitante = solicitud->solicitante;
    printf( format: "Tienes una solicitud de amistad de %s\n",solicitante);
    printf( format: "1. Aceptar solicitud\n"
    "2. Rechazar solicitud\n"
    "3. Ver siguiente solicitud\n"
    "4. Volver al menu de usuario\n"
    "Elige una opcion: ");

    int opcion;
    scanf( format: "%d", &opcion);
    switch (opcion){
        case 1:
            acceptFriendRequest(usuario, solicitante);
            processFriendRequests(usuario);
            break;
        case 2:
            rejectFriendRequest(usuario,solicitante);
            processFriendRequests(usuario);
        case 3:
            if(solicitud==queue->ultimo){
                printf( format: "\nNo tienes mas solicitudes");
            }
            solicitud=solicitud->siguiente;
        case 4:
            operusermenu( user: usuario);

        default:
            printf( format: "Opcion no valida, intente de nuevo\n");
            break;
    }
```

Funció processFriendRequest

En el cas de refusar la sol·licitud d'amistat simplement s'elimina la Request de la RequestQueue però per fer això primer hem de veure si és primer, si és l'últim o si no és cap dels anteriors cassos. En el primer cas simplement s'estableix el següent request com a primer. Els dos següents casos es poden solucionar de la mateixa manera, aquesta seria recórrer tota la RequestQueue request per Request fins a arribar al Request(PrevRequest) que tingui com a següent la Request que estem operant. Un cop el tenim el PrevRequest simplement establim el següent Request del PrevRequest com el següent Request del Request que volem eliminar. I en el cas de que el que volem eliminar sigui l'últim Request de la RequestQueue posem el PrevRequest com el nou últim Request(358-383)

```c
void rejectFriendRequest(struct Node* usuario, struct Node* solicitante){
    if(usuario->solicitudes == NULL){
        printf( format: "No tienes solicitudes de amistad pendientes.\n");
        return;
    }

    struct RequestQueue* queue = usuario->solicitudes;
    struct Request* solicitud = queue->primero;
    struct Request* anterior = NULL;

    while(solicitud != NULL){
        if(solicitud->solicitante == solicitante){
            if(anterior == NULL){
                queue->primero = solicitud->siguiente;
            } else {
                anterior->siguiente = solicitud->siguiente;
            }
            free( Memory: solicitud);
            printf( format: "Solicitud de amistad rechazada.\n");
            return;
        }
        anterior = solicitud;
        solicitud = solicitud->siguiente;
    }
    printf( format: "No se encontró la solicitud de amistad de %s.\n", solicitante->nombre);
}
```

Funció RejectFriendRequest

En el cas d'acceptar la sol·licitud d'amistat la part de treure el Request del RequestQueue seria molt similar a la part de refusar o sigui que l'obviarem i només explicarem la part d'afegir el nou Friend en el FriendsQueue recordem que això s'ha de fer tant per sol·licitant cap a receptor com per part de receptor cap a sol·licitant. Primer creem dos variables tipus Friend (Friend1 i Friend2) i li posem el sol·licitant com a variable amigo de Friend1 i el receptor al del Friend2. Un cop inicialitzades les variables es comprova si el solicitant te Friends, si té s'estableix tant el primer com l'últim Friend com Friend2. Si no se establece Friend2 com l'ultimo i el que era el siguiente Friend del ultimo de la FriendsQueue és Friend2. En el cas del receptor es fa el mateix però amb,b Friend1.(331-353)

```
Amigo->amigo=solicitante;
Amigo->next=NULL;
Amigo2->amigo=receptor;
Amigo2->next=NULL;
if (actualuser->solicitante==solicitante){
    receptor->solicitudes->primero=actualuser->siguiente;
    if(solicitante->Amigos->primero==NULL){

        solicitante->Amigos->primero=Amigo2;
        solicitante->Amigos->ultimo=Amigo2;

    }
    else {
        solicitante->Amigos->ultimo->next = Amigo2;
        solicitante->Amigos->ultimo = solicitante->Amigos->ultimo->next;
    }
    if(receptor->Amigos->primero==NULL){

        receptor->Amigos->primero=Amigo;
        receptor->Amigos->ultimo=Amigo;
        return;
    }
    receptor->Amigos->ultimo->next=Amigo;
    receptor->Amigos->ultimo=receptor->Amigos->ultimo->next;
    return;


}
```

Una altra cosa que ens demanaven es quedessin una funció que enviessin tres sol·licituds d'amistats a tres persones aleatòries. Per això vam fer una funció que creés tres nombres aleatoris entre 1 i el número total de Nodes mirem quin és el Node amb aquest id i si aquest no és Friend o ell mateix llavors es crida a la funció addfriend.(449-463)

Funció revisallista

```
void searchrandfriends(struct Node* solicitante ){
    int i=0, k;
    struct Node* actualuser;
    while(i<3){
        k=1+rand()%(usuarios.last->num);
        actualuser=usuarios.first;
        for(int j=1;j<k;j++){
            actualuser=actualuser->siguiente;
        }
        if(actualuser != solicitante && !isfriend( usuario: solicitante, amigo: actualuser)){
            addfriend(solicitante, receptor: actualuser);
            i++;
        }
    }
}
```

Searchandfriends

Per revisar si les funcions anteriors anaven bé vam decidir fer una funció que imprimeixi per pantalla tots els amics que tens. Si el primer element de la teva FriendQueue no és NULL llavors imprimeix tots els amics fins que l'actual Friend sigui igual a l'últim Friend de la FriendsQueue.(733-748)

```
void printfriendlist(struct FriendsQueue* Amigos){
    if(Amigos->primero==NULL){
        printf( format: "No tienes amigos");
        return;
    }
    Friend* actualfriend=Amigos->primero;
    int i=1;
    while(actualfriend!=Amigos->ultimo){
        printf( format: "Amigo %d: %s      ID:%d",i,actualfriend->amigo->nombre,actualfriend->amigo->num);
        i++;
        actualfriend=actualfriend->next;
    }
    printf( format: "Amigo %d: %s      ID:%d",i,actualfriend->amigo->nombre,actualfriend->amigo->num);
    return;

}
```

printfriendlist

Un cop feta la part dels Friends i els Requests ara ens toca fer la part de crear i veure les publicacions(Posts). Per això com ja és tradició farem una estructura. En aquest cas l'estructura dels posts seguiría la de les Piles. L'estructura Post aquesta vegada només està formada per una sola estructura la qual conté la publicació i el següent Post i en aquest cas s'emmagatzema el primer Post en l'estructura Node.(57-60)

```
struct Post {
    char contenido[120];
    struct Post* siguiente;
};
```

Estructura Post

Un cop feta els Posts ara podem fer una funció que els vagi creant, per això se li pregunta a l'usuari que vol publicar i un cop introduït el que vol. Comprovem si l'usuari ha publicat una cosa abans, si no es així s'introdueix el que ha escrit l'usuari en una variable Post i es guarda en el seu Node com a primer element. Si en el cas on l'usuari sí que té alguna publicació anem passant al següent Post consecutivament fins que veiem que el següent del Post en el qual estem és NULL en aquest cas posem el nou Post com el següent d'aquest(468-483)

```
void doPost(struct Node* usuario, char* contenido){
    struct Post* nueva = (struct Post*)malloc( Size: sizeof(struct Post));
    strncpy( Dest: nueva->contenido, Source: contenido, Count: 120);
    nueva->contenido[120] = '\0';
    nueva->siguiente = NULL;
    if(usuario->lista == NULL){
        usuario->lista = nueva;
    }
    else {
        struct Post *ultima = usuario->lista;
        while (ultima->siguiente != NULL) {
            ultima = ultima->siguiente;
        }
        ultima->siguiente = nueva;
    }
}
```

Per fer la funció que imprimeix tots els posts d'un usuari primer s'ha de saber l'usuari que volem veure. Per això es demana que s'introdueixi l'id de l'usuario en qüestió, un cop tenim l'id fem servir operuserfound per trobar el Node en qüestió. Un cop tenim el Node anem imprimint tots i cada un dels Posts que aquest ha publicat fins que el Post en qüestió sigui NULL.(487-497)

```c
void revisalista(struct Node* usuario){
    printf( format: "Lista de publicaciones %s: \n", usuario->nombre);
    struct Post* publicacion = usuario->lista;
    while (publicacion != NULL){
        printf( format: "%s\n", publicacion->contenido);
        publicacion= publicacion->siguiente;
    }
}
```

Finalment només ens quedava una cosa obligatòria. Aquesta seria fer una biblioteca que anés contant cada una de les paraules que es publiquen. Per fer aquesta última part se'ns vam pensar a fer un top públic on es puguin veure el top 10 paraules més repetides (sense mostrar el nombre de repeticions) i que hi hagués un compte que agafes el rol d'administrador i que pugues veure el número de vegades que es repeteixen totes les paraules. Per això vàrem crear una estructura (charcount) que contingués una array de strings i una altra de números. On en una es mostrarien les paraules en qüestió i en l'altre el número de vegades que es repeteix (en la posició 1 de l'array de strings estaria la paraula i en la posició 1 dels ints el número de vegades que es repeteix).(38-43)

```c
//Aquest estructura es una Biblioteca on s'emmagatzema tots els posts fets.
struct charcount{
    //post(palabra)
    char post[MAX_LENGTH][150];
    //repeticions del post(conteo)
    int reppost[MAX_LENGTH];
};
```

Charcount

Però perquè aquesta struct sigui igual per a tots vam crear una variable global de tipus charcount(Admincount) que s'inicialitzava nul·la en tots els valors.(104-109)

```c
void Admininicialization(){
    for (int i=0;i<MAX_LENGTH;i++) {
        memset( Dst: Admincount->post[i], Val: 0, Size: sizeof(Admincount->post[i]));
        Admincount->reppost[i] = 0;
    }
}
```

Inicialització d'adminccount

Però perquè aquesta variable tingués els valors que hagués de tenir vàrem haver d'introduir una nova funció Admincountadd que introduït un string sumes 1 a l'Admincount corresponent d'aquesta paraula i en cas de no estar-hi crear una nova posició. Per comprovar això fem servir un while que passa per tots els elements de l'array de strings fins que sigui igual a NULL i es va comprovant lletra per lletra si és la mateixa paraula si una lletra és diferent doncs passem al següent. Si la paraula no està anem al primer element NULL de l'array i el substituïm pel string i en aquella mateixa posició de l'array d'ints s'iguala a 1. Si la paraula està li sumem u a la posició de l'array d'integers(752-809)

```
void Admincountadd(char post[150]){
    int i=0,eq=TRUE,j;
    while(Admincount->reppost[i]!=0&&i<MAX_LENGTH){
        j=0;
        eq=TRUE;
        while(post[j]!=0&&eq==TRUE){
            if(post[j]!=Admincount->post[i][j]){
                eq=FALSE;
            }
            j++;

        }
        if(eq==TRUE) {
            Admincount->reppost[i]++;
            if (i == 0) {
                return;

            }
            while (Admincount->reppost[i] > Admincount->reppost[i - 1]){


                std::swap( &Admincount->reppost[i], &Admincount->reppost[i - 1]);

                std::swap( &Admincount->post[i], &Admincount->post[i - 1]);

                i--;
                if (i == 0) {
                    return;
                }
            }
            return;
        }
        i++;
    }

    j=0;
    while(post[j]!=0){
        Admincount->post[i][j]=post[j];
```

```
        j++;
    }

    Admincount->reppost[i]=1;
    return;
}
```

funció Admincountadd

Ara simplement ens queda una funció que imprimeixi les deu paraules més utilitzades i una altra que imprimeixi totes les paraules i el número de vegades que surten. Com totes dues funcionen exactament de la mateixa manera ho explicaré en conjunt. Fem un bucle que es vagi repetint fins que l'element de l'array d'integers en el que ens trobem sigui igual a zero. i simplement fem el print dels elements que es troben en aquella posició de l'array(924-939)+(487-494)

```
while(Admincount->reppost[i]!=0){
    printf( format: "'%s' se ha repetido %d ",Admincount->post[i],Admincount->reppost[i]);
    if(Admincount->reppost[i]==1){
        printf( format: "vez.\n");
    }
    else{
        printf( format: "veces.\n");
    }
    i++;

}
```

En el nostre cas vam posar que l'administrador fos el primer usuari creat.

Per acabar vàrem fer una funcionalitat extra d'implementar una contrasenya en el compte d'usuari. Per això vàrem afegir dos elements en l'estructura Node, l'ispass(bool) que comença sent FALSE i el password(string). L'ispass t'indica si el compte té contrasenya i el password és la contrasenya. Per això vàrem fer una opció més en l'opermenu la qual fos canviar la contrasenya. Si selecciones se't pregunta quina vols que sigui la teva nova contrasenya i aquesta passa a ser el password del Node en qüestió i l'ispass passa a ser TRUE.(905-916)

```
case 8:
    printf( format: "Seleccion: Cambio de password\n");
    user->ispass=TRUE;
    printf( format: "Introduce la password que desea tener\n");
    char pass[MAX_LENGTH];
    scanf( format: "%s",pass);
    int i;
    i = 0;
    while(pass[i]!=0){
        user->password[i]=pass[i];
        i++;
    }
    break;
```

Un cop creada una contrasenya ara en accedir a l'operusermenu() des del menu() se t'advertirà que el compte està protegit i se't preguntarà la contrasenya. Si acceptes a introduir la contrasenya es comprovarà mitjançant un bucle lletra per lletra si la contrasenya introduïda és igual a password si és així podràs entrar a operusermenu().(954-1016)

```
bool checkpass(char pass[MAX_LENGTH],struct Node* user){
    int i=0;
    while(user->password[i]!=0){
        if(pass[i]!=user->password[i]){
            return FALSE;
        }
        i++;
    }
    if(pass[i]!=0){
        return FALSE;
    }
    return TRUE;

}
```

La funció "suggestFriends" rep un usuari com a paràmetre i suggereix possibles amics per a aquest usuari basant-se en els seus gustos musicals. Recorre tots els usuaris, buscant aquells que comparteixin gèneres musicals amb l'usuari donat. Si es troben suficients

coincidències, aquests usuaris es consideren suggeriments d'amistat. Finalment, es mostren els suggeriments per pantalla. Si no es troben suggeriments, es mostra un missatge indicant-ho.

```
void suggestFriends(struct Node* usuario) {
    // Se Crea una lista para almacenar las sugerencias de amigos
    struct Node* sugerencias[MAX_LENGTH];
    int numSugerencias = 0;

    // Se recorren a todos los usuarios y busca aquellos que tengan géneros en común.
    struct Node* aux = operuserfound( i: 1); //
    while (aux != NULL) {
        // Verifica si el usuario actual no es el mismo y no es amigo del usuario actual
        if (aux != usuario && !isfriend(usuario, amigo: aux)) {
            // Verifica si hay géneros de música en común
            int numGustosComunes = 0;
            for (int i = 0; i < usuario->num; i++) {
                for (int j = 0; j < aux->num; j++) {
                    if (strcmp(usuario->musica[i], aux->musica[j]) == 0) {
                        numGustosComunes++;
                        break;
                    }
                }
            }

            // Agregar el usuario a la lista de sugerencias si hay suficientes gustos en común
            if (numGustosComunes >= 2) {
                sugerencias[numSugerencias] = aux;
                numSugerencias++;
            }
        }

        aux = aux->siguiente;
    }

    // Se muestran las sugerencias de amigos
    printf( format: "Sugerencias de amigos para %s:\n", usuario->nombre);
    if (numSugerencias > 0) {
```

```
        for (int i = 0; i < numSugerencias; i++) {
            printf( format: "%d. %s\n", i + 1, sugerencias[i]->nombre);
        }
    } else {
        printf( format: "No se encontraron sugerencias de amigos.\n");
    }
}
```

Els objectius obligatoris s'han fet parcialment, hem fet ús de la llista i la cua en exemples com les estructures utilitzades.

L'estructura "Arraynode" no representa directament una llista, pila o cua. En canvi, emmagatzema els punters al primer i últim usuari guardat, el qual es pot utilitzar per recórrer tots els usuaris en un cert ordre.

L'estructura "Friend" té un punter a "Node", que representa un amic, i un punter a "Friend", que representa l'amic següent en una llista enllaçada. Aquesta estructura s'assembla a una llista enllaçada d'amics, on cada amic es connecta amb el següent mitjançant el punter "next".

L'estructura "FriendsQueue" també té punters a "Friend", que representen el primer i l'últim amic a la cua d'amics. Encara que se li diu "cua d'amics", no s'implementa com una cua tradicional (FIFO). En canvi, sembla ser una llista enllaçada d'amics, similar a l'estructura "Friend".

L'estructura "Post" representa una publicació i utilitza un punter a "Post" per representar la següent publicació en una llista enllaçada. En aquest cas, l'estructura "Post" es pot considerar com una llista enllaçada de publicacions, on cada publicació es connecta amb la següent mitjançant el punter "següent".

A Objectius Desitjables s'han completat la de llegir dades d'una font com un arxiu de text, aquesta es troba a initfuser, la qual ja s'ha comentat prèviament.
Per altra banda, la temàtica de la xarxa social té un personalització orientada cap als gèneres de música. A més hi haurà un apartat de suggeriments per gent a la que li agradi els gèneres que tenen en comú.

SOLUCIÓ

Arquitectura del sistema

L'arquitectura del sistema consta de varios components principals que interactuen entre si per aconseguir el funcionament desitjat.

El mòdul d'usuaris: En aquest es gestiona la informació dels usuaris, en aquest s'inclouen nom, contrasenya, edat, correu electrònic, ciutat, música, llista d'amics etc. Utilitza l'estructura de dades Node per representar cada usuari.

El mòdul de publicacions: Es gestionen les publicacions realitzades pels usuaris, utilitza l'estructura de dades Post, que conté el contingut de la publicació i un punter al següent post a una llista enllaçada.

Mòdul de sol·licituds: Aquest s'encarrega de gestionar les sol·licituds d'amistat entre els usuaris. Utilitza l'estructura de dades Request, on es guarden els sol·licitants i el receptor de la solicitud, a més d'un punter del següent element a una llista enllaçada.

Mòdul de conteig de caràcters: S'utilitza per realitzar un conteig de caràcters a les publicaciones. Utilitza l'estructura de dades charcount, que emmagatzema les publicacions i el número de repeticions de cadascuna.

Gestió d'errors

Els errors que s'han trobat han sigut a funcions com initfuser, en la que s'obre un arxiu, però no se sabia com obrir-se. Per altra banda, el charcount hauria d'haver sigut un dynamic array, ja que així no es pot omplir.

A la gestió d'amistats, no es pot visualitzar les sol·licituds anteriors, per tant, s'ha de tornar al menú.
Finalment a l'hora imprimir els post, únicament s'imprimeix una paraula i '\n', no permet imprimir-se la frase completa.

Per altra banda, per poder evitar altres errors, el programa implementa mecanismes per gestionar situacions inesperades o incorrectes.

Un exemple pot ser que les dades introduïdes pels usuaris compleixen certs criteris, com tenir una longitud màxima. A més de l'ús de if i bucles per verificar i poder veure si són vàlides una sèrie de dades.

Disseny del model de dades

El model de dades inclou estructures Node, Sol·licitud, Publicació, recompte de dades i altres estructures. Aquestes estructures contenen propietats que representen informació sobre cada entitat.
Node: emmagatzema informació sobre els usuaris, com ara els seus noms, contrasenyes i llistes d'amics.
Sol·licitud: informació sobre les sol·licituds d'amistat, inclosos el sol·licitant i el receptor.
Publicació: emmagatzema el contingut de la publicació i un punter a la publicació següent.
Charcount s'utilitza per calcular i emmagatzemar informació.

Descripció i processament del conjunt de dades

El sistema pot interactuar amb les dades de l'usuari, les publicacions i les sol·licituds introduïdes a través de la interfície d'usuari.
El processament de conjunts de dades pot implicar llegir l'entrada de l'usuari i emmagatzemar-la a les estructures de dades corresponents, com ara Node per a usuaris, Publicació per a publicacions i Sol·licitud de sol·licitud d'amistat.
A més, es poden realitzar operacions de validació i validació de les dades introduïdes per assegurar-se que compleixen determinats requisits, com ara la longitud màxima d'un camp de text o la presència de caràcters vàlids.

En resum, el sistema interactua amb conjunts de dades subministrats per l'usuari que es processen i s'emmagatzemen en estructures de dades adequades per a un ús posterior i funcional dins del sistema.

REFERENCIAS

En aquest treball s'ha fet ús de git-hub, chat pgt ,'lawebdelprogamador' i Youtube