	**Primer Proyecto de EDA I**
Facultad de Ingeniería	Streaming de Música

Primer Proyecto de EDA I

Profesor: Ing. Patricia del Valle Morales

Asignatura: Estructura de Datos y Algoritmos 1 (EDA I)

Grupo: 2

Integrante(s): Gómez Martínez Cristopher Emiliano
Lugo Sáenz Jesús
Nolasco Sotelo Brenda Carolina
Ordiales Caballero Iñaky

Semestre: 2020-2

Fecha de entrega: 29 /Marzo/2020

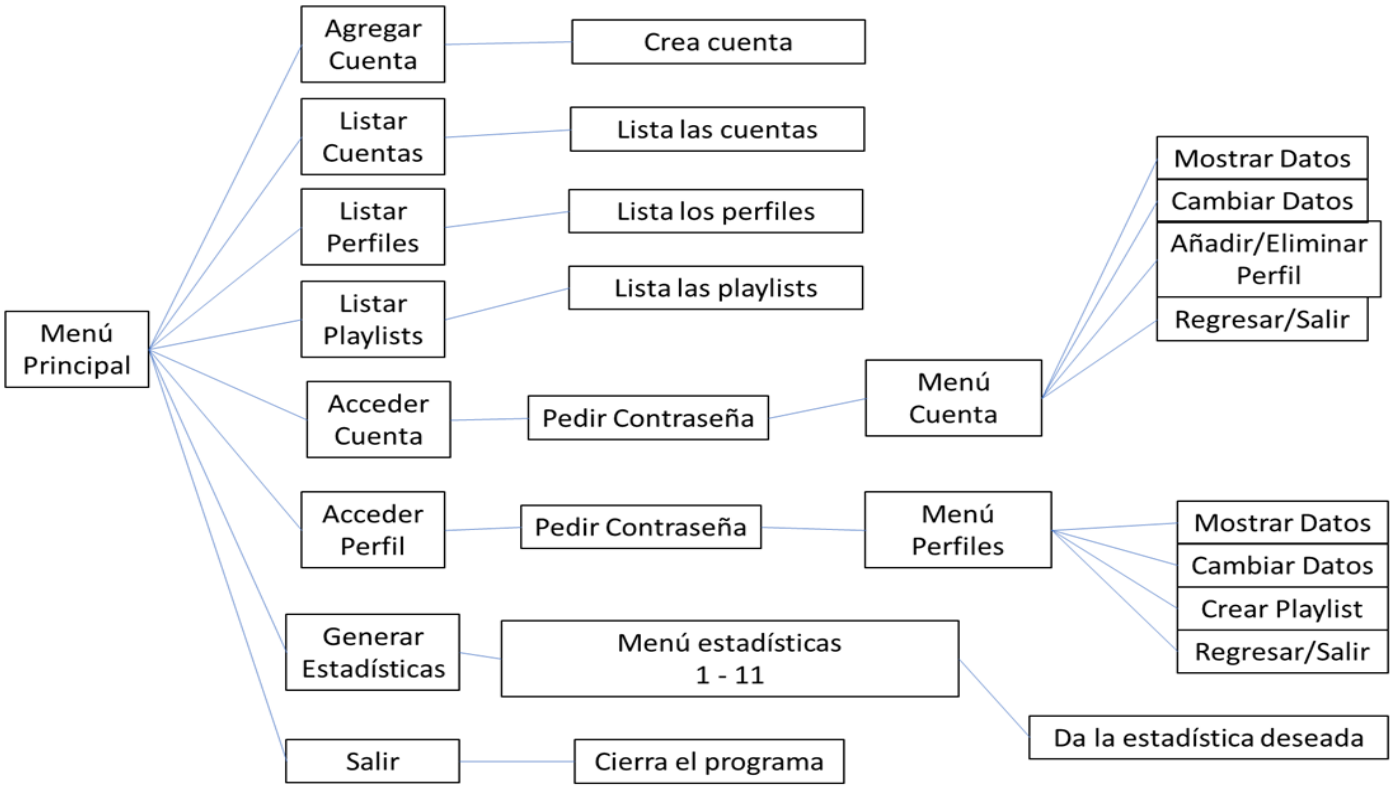
Observaciones:

CALIFICACIÓN: _____

Proyecto 1

Plataforma Streaming de música

Para la elaboración de nuestro proyecto nosotros decidimos hacer una plataforma de reproducción de música. En ella tiene funcionalidades de cuentas y perfiles, donde se pueden ver los playlists (listas de reproducción) hechos por otros usuarios o crear uno. Además las cuentas van desde gratis hasta con precio, dependiendo de lo que se desee. La estructura básica del programa se muestra en el siguiente diagrama:



El programa originalmente se pensó para música, claro con las limitaciones de ser un prototipo que simula una, pero que realmente no puede reproducir canciones. Sin embargo se trató de hacerla lo más apegada a la realidad por lo que por ejemplo: te pide confirmar contraseña al registrarte y si no coinciden te la rechaza, no puede haber dos usuarios con el mismo correo electrónico entonces lo compara con todos los usuarios anteriores y también te lo rechaza si es repetido. Al acceder a una cuenta o perfil te pide el correo y la contraseña y si no coinciden te rechaza.

A pesar de ser un programa para música fácilmente se podría usar como plantilla y adecuar a otras cosas, por ejemplo un censo donde por ejemplo en lugar de cuentas serían casas, en lugar de perfiles serían las personas que viven en la casa y las playlists serían las posesiones de la persona. Ya se tendría la estructuración hecha.

Finalmente a continuación se muestran los requerimientos del programa.

1. Tipo de datos estructuras (struct) compuestas o anidadas con apuntadores

```c
typedef struct{
    char nombre[30];
    char email[30];
    char contrasena[30];
    int edad;
    int dia;
    int mes;
    int ano;
    char genero[30];
    int numCuenta;
    int numPerfil;
} Perfil;

typedef struct{
    char calle[30];
    int numeroDomicilio;
    char colonia[30];
    int CP;
    char ciudad[30];
    char pais[30];
} Domicilio;

typedef struct{
    char formaPago[30];
    char numeroTarjeta[30];
    int mesVencimiento;
    int anoVencimiento;
    int CVV;
    float precio;
} Pago;

typedef struct{
    char paquete[30];
    Pago *formaPago;
} Plan;

typedef struct{
    Perfil *duenoCuenta;
    Domicilio *domicilio;
    Plan *tipoPlan;
    int *perfiles;
    int numCuenta;
} Cuenta;

typedef struct{
    char *nombre;
    char *artista;
    int lanzada;
}Cancion;

typedef struct{
    char *nombrePlaylist;
    int numeroCanciones;
    char *creador;
    int descargas;
    Cancion *canciones;
} Playlist;
```

Se declararon las estructuras, las cuales estaban anidadas dentro de la estructura cuenta, todas las estructuras anidadas se crean dinámicamente y algunas de las variables de las estructuras.

2. Estructura de datos lineal llamada arreglo, creado dinámicamente.

```
18      Cuenta *cuentas;
19      Perfil *perfiles;
20      Playlist *playlists;
21
22      cuentas = (Cuenta *) malloc(MAXcuentas*sizeof(Cuenta));
23      if(cuentas == NULL){
24          printf("Error al crear cuentas...");
25          return -1;
26      }
27      perfiles = (Perfil *) malloc(MAXperfiles*sizeof(Perfil));
28      if(perfiles == NULL){
29          printf("Error al crear perfiles...");
30          return -1;
31      }
32      playlists = (Playlist *) malloc(MAXplaylists*sizeof(Playlist));
33      if(playlists == NULL){
34          printf("Error al crear playlists...");
35          return -1;
36      }
```

En la función main se crean tres arreglos lineales de estructura dinámicamente al iniciarse el programa. Como se observa en el código de arriba, se crean los tres arreglos de tamaño Cuenta, Perfil y Playlist respectivamente. La creación se da con el número de elementos establecido por MAXcuentas, MAXperfiles y MAXplaylists, por lo que, si se quisiera crecer la capacidad de almacenamiento del programa, tan solo se deben cambiar esas constantes. Si no se pudieran crear los arreglos, manda mensaje de error y cierra el programa con -1.

3. Memoria dinámica (malloc, calloc, realloc, free)

```
void crearCuenta(Cuenta *nuevaCuenta){

    nuevaCuenta->duenoCuenta = (Perfil *) malloc(sizeof(Perfil));
    if (nuevaCuenta == NULL){
        printf("ERROR al crear perfil");
        exit(-1);
    }

    nuevaCuenta->domicilio = (Domicilio *) malloc(sizeof(Domicilio));
    if (nuevaCuenta == NULL){
        printf("ERROR al crear domicilio");
        exit(-1);
    }

    nuevaCuenta->tipoPlan = (Plan *) malloc(sizeof(Plan));
    if (nuevaCuenta == NULL){
        printf("ERROR al crear plan");
        exit(-1);
    }

    nuevaCuenta->tipoPlan->formaPago = (Pago *) malloc(sizeof(Pago));
    if (nuevaCuenta == NULL){
        printf("ERROR al crear pago");
        exit(-1);
    }

    nuevaCuenta->perfiles = (int *) malloc(5*sizeof(int));
    if (nuevaCuenta == NULL){
        printf("ERROR al crear plan");
        exit(-1);
    }
}
```

```c
void crearPlaylist(Playlist *nuevaPlaylist){
    nuevaPlaylist->creador = (char *)malloc(30*sizeof(char));
    if (nuevaPlaylist->creador == NULL){
        printf("Error al crear creador playlist...");
        exit(-1);
    }
    nuevaPlaylist->nombrePlaylist = (char *)malloc(30*sizeof(char));
    if (nuevaPlaylist->nombrePlaylist == NULL){
        printf("Error al crear nombre playlist...");
        exit(-1);
    }
}

Cancion *crearCanciones(int numCanciones){
    int i;
    Cancion *canciones;

    canciones = (Cancion *) calloc(numCanciones, sizeof(Cancion));
    if (canciones == NULL){
        printf("Error al crear canciones");
        exit(-1);
    }
    for (i=0; i<numCanciones; i++){
        canciones[i].nombre = (char *) calloc(30, sizeof(char));
        if (canciones[i].nombre == NULL){
            printf("Error al crear nombre cancion");
            exit(-1);
        }
        canciones[i].artista = (char *) calloc(30, sizeof(char));
        if (canciones[i].artista == NULL){
            printf("Error al crear nombre artista");
            exit(-1);
        }
    }
    return canciones;
}
```

Se crearon las estructuras de forma dinámica usando las funciones “malloc” y “calloc” y se implementó una función “if” con la cual se verificó que su valor fuera diferente de “NULL”.

```c
707                     switch(tipo){
708                         case 3:
709                             cuenta->tipoPlan->formaPago->precio = 99;
710                             realloc(cuenta->duenoCuenta, 1*sizeof(Perfil));
711                             cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
712                             break;
713
714                         case 4:
715                             cuenta->tipoPlan->formaPago->precio = 129;
716                             realloc(cuenta->duenoCuenta, 2*sizeof(Perfil));
717                             cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
718                             break;
719                         case 5:
720                             cuenta->tipoPlan->formaPago->precio = 149;
721                             realloc(cuenta->duenoCuenta, 6*sizeof(Perfil));
722                             cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
723                             break;
724                         case 6:
725                             cuenta->tipoPlan->formaPago->precio = 49;
726                             realloc(cuenta->duenoCuenta, 1*sizeof(Perfil));
727                             cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
728                             break;
```

Además durante la ejecución dependiendo del plan que la cuenta elija, se usa la función realloc para el arreglo de enteros de Cuenta, el cual contiene los números de Perfil asociados a la cuenta.

```c
void liberarMDCuenta(Cuenta *unacuenta){
    free(unacuenta->perfiles);
    free(unacuenta->tipoPlan->formaPago);
    free(unacuenta->tipoPlan);
    free(unacuenta->domicilio);
    free(unacuenta->duenoCuenta);
    free(unacuenta);

    unacuenta = NULL;
}

void liberarMDPerfil(Perfil *unperfil){
    free(unperfil);
    unperfil = NULL;
}

void liberarMDPlaylist(Playlist *unaplaylist){
    free(unaplaylist->creador);
    free(unaplaylist->nombrePlaylist);
    liberarMDCancion(unaplaylist->canciones, unaplaylist->numeroCanciones);
    free(unaplaylist);
    unaplaylist = NULL;
}

void liberarMDCancion(Cancion *canciones, int numCanciones){
    int i;
    for(i=0; i<numCanciones; i++){
        free(canciones[i].nombre);
        free(canciones[i].artista);
    }
    free(canciones);
}

void comprimir(char *frase){
    int i;
    for (i=0; i<strlen(frase); i++){
        if(frase[i] == ' ')
            frase[i] = '}';
    }
}

void descomprimir(char *frase){
    int i;
    for (i=0; i<strlen(frase); i++){
        if(frase[i] == '}')
            frase[i] = ' ';
    }
}
```

Código del main:

```c
219         for (i=0; i<=iCuentas; i++)
220             liberarMDCuenta(&cuentas[i]);
221
222         for (i=0; i<=iPerfiles; i++)
223             liberarMDPerfil(&perfiles[i]);
224
225         for (i=0; i<=iPlaylists; i++){
226             liberarMDPlaylist(&playlists[i]);
227         }
```

Al finalizar el programa se liberan las estructuras y arreglos que fueron creadas con memoria dinámica con la función “free” y si igualan a “NULL” para que no se puedan ocupar una vez cerrado el programa

4. Todo deberá de estar implementado con funciones con parámetros por valor y por referencia (apuntadores)

```
int extraerContadores(int *iCuentas, int *iPerfiles, int *iPlaylists);
int guardarContadores(int iCuentas, int iPerfiles, int iPlaylists);

int bajarArchivoCuentas(Cuenta *cuentas, int iCuentas);
int bajarArchivoPerfiles(Perfil *perfiles, int iPerfiles);
int bajarArchivoPlaylists(Playlist *playlists, int iPlaylists);

void crearCuenta(Cuenta *nuevaCuenta);
void crearPlaylist(Playlist *nuevaPlaylist);
Cancion *crearCanciones(int numCanciones);

int validarEspacioCuentas(int iCuentas, int MAXcuentas);
int validarEspacioPerfiles(int iPerfiles, int MAXperfiles);
int validarEspacioPlaylists(int iPlaylists, int MAXplaylists);

int comprobarCorreo(Perfil *perfiles, int iPerfiles);

void capturarCuenta(Cuenta *unacuenta, Perfil *perfiles, int *iPerfiles);
void capturarPerfil(Perfil *perfiles, int *iPerfiles, int numCuenta);
void capturarPlaylist(Playlist *playlist, char creador[]);

void listarCuenta(Cuenta cuenta, Perfil *perfiles);
void listarPerfil(Perfil perfil);
void listarPlaylist(Playlist playlist);

int buscarCuenta(Cuenta *cuentas, char correo[], int iCuentas);
int buscarPerfil(Perfil *perfiles, char correo[], int iPerfiles);

int checarContrasenaCuenta(Cuenta *cuentas, int iCuentas, int *numCuenta);
int checarContrasenaPerfil(Perfil *perfiles, int iPerfiles, int *numPerfil);

void cambiarDatosCuenta(Cuenta *cuenta, Perfil *perfiles, int iPerfiles);
void cambiarDatosPerfil(Perfil *perfil, Perfil *perfiles, int iPerfiles);
void borrarPerfil(Cuenta *cuenta, Perfil *perfiles, int *iPerfiles, int Borrar);
void gestionarPerfilesCuenta(Cuenta *cuentas, int numCuenta, Perfil *perfiles, int *iPerfiles);

void Estad_1_porcentajeHombresYMujeres(Perfil *perfiles, int iPerfiles);
void Estad_2_porcentajeCuentas(Cuenta *cuentas, int iCuentas);
void Estad_3_playlistsRegistradas(int iPlaylists);
void Estad_4_ingresosRegistrados(Cuenta *cuentas, int iCuentas);
void Estad_5_metodoPago(Cuenta *cuentas,int iCuentas);
void Estad_6_edadPromedio(Perfil *perfiles,int iPerfiles);
void Estad_7_usuariosRegistrados(int iPerfiles);
void Estad_8_cuentaMasCostosa(Cuenta *cuentas, int iCuentas);
void Estad_9_playlistsReistrados(Playlist *playlists, int iPlaylists);
void Estad_10_masCanciones(Playlist *playlists, int iPlaylists);
void Estad_11_epoca(Playlist *playlists, int iPlaylists);

int subirArchivoCuentas(Cuenta *cuentas, int iCuentas);
int subirArchivoPerfiles(Perfil *perfiles, int iPerfiles);
int subirArchivoPlaylists(Playlist *playlists, int iPlaylists);

void liberarMDCuenta(Cuenta *cuenta);
void liberarMDPerfil(Perfil *perfil);
void liberarMDPlaylist(Playlist *playlist);
void liberarMDCancion(Cancion *canciones, int numCanciones);

void comprimir(char *frase);
void descomprimir(char *frase);
```

Declaramos todos los prototipos de las funciones con sus respectivos valores a recibir de acuerdo al propósito de cada una de las funciones. Se pasan la mayoría de valores por referencia y si no por copia, no se utilizaron variables globales en ninguna parte.

Código del main:

```
case 1:
    if(validarEspacioCuentas(iCuentas, MAXcuentas) && validarEspacioPerfiles(iPerfiles+5, MAXperfiles)){
        iCuentas++;
        crearCuenta(&cuentas[iCuentas]);
        cuentas[iCuentas].numCuenta = iCuentas;
        capturarCuenta(&(cuentas[iCuentas]), perfiles, &iPerfiles);
    }else
        printf("\n\n Ya no hay espacio para mas cuentas o perfiles en el servidor.\n");
    break;

case 2:
    if (iCuentas == -1)
        printf("\n No hay cuentas existentes.\n");
    else{
        for (i=0; i<=iCuentas; i++)
            listarCuenta(cuentas[i], perfiles);
    }
    break;

case 3:
    if (iPerfiles == -1)
        printf("\n No hay perfiles existentes.\n");
    else{
        for (i=0; i<=iPerfiles; i++)
            listarPerfil(perfiles[i]);
    }
    break;

case 4:
    if (iPlaylists == -1)
        printf("\n No hay playlists existentes.\n");
    else{
        for (i=0; i<=iPlaylists; i++)
            listarPlaylist(playlists[i]);
    }
    break;

case 5:
    if (checarContrasenaCuenta(&cuentas[0], iCuentas, &numCuenta)){
        do{
            opcion = Menu("\n - Menu Cuenta - \n\t1)Mostrar datos\n\t2)Cambiar datos\n\t3)Administrar perfiles\n\t4)Regresar\n Opcion: ", 4);
            switch (opcion){
                case 1:
                    listarCuenta(cuentas[numCuenta], perfiles);
                    break;

                case 2:
                    cambiarDatosCuenta(&cuentas[numCuenta], perfiles, iPerfiles);
                    break;

                case 3:
                    gestionarPerfilesCuenta(&cuentas[numCuenta], numCuenta, perfiles, &iPerfiles);
                    break;
            }
        }while (opcion !=4);
    }
    break;
```

```
case 6:
    if (checarContrasenaPerfil(&perfiles[0], iPerfiles, &numPerfil)){
        do{
            opcion = Menu("\n - Menu Perfil - \n\t1)Mostrar datos\n\t2)Cambiar datos\n\t3)Crear Playlist\n\t4)Regresar\n Opcion: ", 4);
            switch (opcion){
                case 1:
                    listarPerfil(perfiles[numPerfil]);
                    break;

                case 2:
                    cambiarDatosPerfil(&perfiles[numPerfil], perfiles, iPerfiles);
                    break;

                case 3:
                    if(validarEspacioPlaylists(iPlaylists, MAXplaylists)){
                        iPlaylists++;
                        capturarPlaylist(&playlists[iPlaylists], perfiles[numPerfil].email);
                    }else
                        printf("\n No hay espacio en el servidor para crear mas playlists.\n");
                    break;
            }
```

```
case 1:
    if (iPerfiles>-1)
        Estad_1_porcentajeHombresYMujeres(perfiles,iPerfiles);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 2:
    if (iCuentas>-1)
        Estad_2_porcentajeCuentas(cuentas,iCuentas);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 3:
    if (iPlaylists>-1)
        Estad_3_playlistsRegistradas(iPlaylists);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 4:
    if (iCuentas>-1)
        Estad_4_ingresosRegistrados(cuentas,iCuentas);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 5:
    if (iCuentas>-1)
        Estad_5_metodoPago(cuentas, iCuentas);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 6:
    if (iPerfiles>-1)
        Estad_6_edadPromedio(perfiles, iPerfiles);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 7:
    if (iPerfiles>-1)
        Estad_7_usuariosRegistrados(iPerfiles);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 8:
    if (iCuentas>-1)
        Estad_8_cuentaMasCostosa(cuentas,iCuentas);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 9:
    if (iPlaylists>-1)
        Estad_9_playlistsReistrados(playlists, iPlaylists);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 10:
    if (iPlaylists>-1)
        Estad_10_masCanciones(playlists, iPlaylists);
    else
        printf("\n No hay datos registrados...\n");
    break;
case 11:
    if (iPlaylists>-1)
        Estad_11_epoca(playlists, iPlaylists);
    else
        printf("\n No hay datos registrados...\n");
    break;
```

En el main de acuerdo a la función se pasaban los datos ya sea por referencia o por copia.

5. Todas las operaciones sobre un arreglo: a. Insertar b. Extraer o borrar c. Actualizar datos d. Buscar un dato en particular e imprimirlo en pantalla e. Listar toda la información almacenada en las estructuras lineales "arreglo"

Funciones prototipo:

```
void capturarCuenta(Cuenta *unacuenta, Perfil *perfiles, int *iPerfiles);
void capturarPerfil(Perfil *perfiles, int *iPerfiles, int numCuenta);
void capturarPlaylist(Playlist *playlist, char creador[]);

void listarCuenta(Cuenta cuenta, Perfil *perfiles);
void listarPerfil(Perfil perfil);
void listarPlaylist(Playlist playlist);

int buscarCuenta(Cuenta *cuentas, char correo[], int iCuentas);
int buscarPerfil(Perfil *perfiles, char correo[], int iPerfiles);

void cambiarDatosCuenta(Cuenta *cuenta, Perfil *perfiles, int iPerfiles);
void cambiarDatosPerfil(Perfil *perfil, Perfil *perfiles, int iPerfiles);
void borrarPerfil(Cuenta *cuenta, Perfil *perfiles, int *iPerfiles, int Borrar);
```

a. Insertar

```
void capturarPerfil(Perfil *perfiles, int *iPerfiles, int numCuenta){
    int aux;
    char contra_1[30], contra_2[30];
    char generos[3][12] = {"Masculino", "Femenino", "No-Binario"};

    *iPerfiles += 1;

    perfiles[*iPerfiles].numCuenta = numCuenta;
    perfiles[*iPerfiles].numPerfil = *iPerfiles;

    printf("\nDATOS PERSONA.\n");
    printf("Nombre: ");
    fflush(stdin);
    gets(perfiles[*iPerfiles].nombre);
    printf("Email: ");
    fflush(stdin);
    gets(perfiles[*iPerfiles].email);
    while (!comprobarCorreo(perfiles, *iPerfiles)){
        printf("\nEse correo ya esta utilizado!!!\nPor favor ingrese uno nuevo.\nEmail: ");
        fflush(stdin);
        gets(perfiles[*iPerfiles].email);
    }
    printf("Contrasena: ");
    fflush(stdin);
    gets(contra_1);
    printf("Comprobar contrasena: ");
    fflush(stdin);
    gets(contra_2);
    while (strcmp(contra_1, contra_2)){
        printf("Las contrasenas no coinciden!!!\n");
        printf("Vuelva a ingresar contrasena: ");
        fflush(stdin);
        gets(contra_1);
        printf("Vuelva a comprobar contrasena: ");
        fflush(stdin);
        gets(contra_2);
    }
    strcpy(perfiles[*iPerfiles].contrasena, contra_1);
    printf("Edad: ");
    scanf("%i", &perfiles[*iPerfiles].edad);
    printf("Dia de nacimiento (nn): ");
    scanf("%i", &perfiles[*iPerfiles].dia);
    printf("Mes de nacimiento (nn): ");
    scanf("%i", &perfiles[*iPerfiles].mes);
    printf("Ano de nacimiento (nnnn): ");
    scanf("%i", &perfiles[*iPerfiles].ano);
    printf("Genero. [1]Masculino    [2]Femenino     [3]No Binario\nOpcion: ");
    scanf("%i", &aux);
    while (aux<1 || aux>3){
        printf("Opcion no valida, ingrese otra.\nOpcion: ");
        scanf("%i", &aux);
    }
    strcpy(perfiles[*iPerfiles].genero, generos[aux-1]);
}
```

Esta función sirve para capturar los datos de la estructura Perfil e insertarlo en el arreglo de perfiles (arreglo de estructuras Perfil) ,checando que el correo electrónico no se vuelva a repetir dentro del arreglo.

b. Borrar

```
void borrarPerfil(Cuenta *cuenta, Perfil *perfiles, int *iPerfiles, int iBorrar){

    strcpy(perfiles[cuenta->perfiles[iBorrar]].nombre, perfiles[*iPerfiles].nombre);
    strcpy(perfiles[cuenta->perfiles[iBorrar]].email, perfiles[*iPerfiles].email);
    strcpy(perfiles[cuenta->perfiles[iBorrar]].contrasena, perfiles[*iPerfiles].contrasena);
    perfiles[cuenta->perfiles[iBorrar]].edad = perfiles[*iPerfiles].edad;
    perfiles[cuenta->perfiles[iBorrar]].dia = perfiles[*iPerfiles].dia;
    perfiles[cuenta->perfiles[iBorrar]].mes = perfiles[*iPerfiles].mes;
    perfiles[cuenta->perfiles[iBorrar]].ano = perfiles[*iPerfiles].ano;
    strcpy(perfiles[cuenta->perfiles[iBorrar]].genero, perfiles[*iPerfiles].genero);
    perfiles[cuenta->perfiles[iBorrar]].numCuenta = perfiles[*iPerfiles].numCuenta;
    cuenta->perfiles[iBorrar] = -1;

    *iPerfiles-=1;

}
```

Esta función se implementa al elegir la opción de borrar el perfil extra de una cuenta. Al seleccionar esa opción, la función borrarPerfil copia los datos del último elemento del arreglo en el que se desea borrar y baja el contador de la última casilla ocupada en una. Eliminando del arreglo ocupado el perfil.

c. Actualizar

Código main:

```
opcion = Menu("\n - Menu Cambio Info Cuenta - \n\t1)Cambiar info dueno de la cuenta \n\t2)Cambiar info domicilio\n\t3)Cambiar i
switch (opcion){
    case 1:
        listarPerfil(*cuenta->duenoCuenta);
        do{
            eleccion = Menu("\n ~ Cambio dueno cuenta ~\n\t1)Nombre\n\t2)Email\n\t3)Contrasena\n\t4)Edad\n\t5)Dia de nacimiento
            switch (eleccion){
                case 1:
                    printf("Nombre: ");
                    fflush(stdin);
                    gets(cuenta->duenoCuenta->nombre);
                    break;

                case 2:
                    printf("Email: ");
                    fflush(stdin);
                    gets(cuenta->duenoCuenta->email);
                    while (!comprobarCorreo(cuenta->duenoCuenta, iPerfiles)){
                        printf("\nEse correo ya esta utilizado!!!\nPor favor ingrese uno nuevo.\nEmail: ");
                        fflush(stdin);
                        gets(cuenta->duenoCuenta->email);
                    }
                    break;

                case 3:
                    printf("Contrasena: ");
                    fflush(stdin);
                    gets(contra_1);
                    printf("Comprobar contrasena: ");
                    fflush(stdin);
                    gets(contra_2);
                    while (strcmp(contra_1, contra_2)){
                        printf("Las contrasenas no coinciden!!!\n");
                        printf("Vuelva a ingresar contrasena: ");
                        fflush(stdin);
                        gets(contra_1);
                        printf("Vuelva a comprobar contrasena: ");
                        fflush(stdin);
                        gets(contra_2);
                    }
                    strcpy(cuenta->duenoCuenta->contrasena, contra_1);
                    break;
```

```
            case 4:
                printf("Edad: ");
                scanf("%i", &cuenta->duenoCuenta->edad);
                break;

            case 5:
                printf("Dia de nacimiento (nn): ");
                scanf("%i", &cuenta->duenoCuenta->dia);
                break;

            case 6:
                printf("Mes de nacimiento (nn): ");
                scanf("%i", &cuenta->duenoCuenta->mes);
                break;

            case 7:
                printf("Ano de nacimiento (nnnn): ");
                scanf("%i", &cuenta->duenoCuenta->ano);
                break;

            case 8:
                printf("Genero. [1]Masculino    [2]Femenino     [3]No Binario\nOpcion: ");
                scanf("%i", &aux);
                while (aux<1 || aux>3){
                    printf("Opcion no valida, ingrese otra.\nOpcion: ");
                    scanf("%i", &aux);
                }
```

```
case 2:
    printf("\nNUEVA DIRECCION\nCalle: ");
    fflush(stdin);
    gets(cuenta->domicilio->calle);
    printf("Numero calle: ");
    scanf("%i", &cuenta->domicilio->numeroDomicilio);
    printf("Colonia: ");
    fflush(stdin);
    gets(cuenta->domicilio->colonia);
    printf("Codigo postal: ");
    scanf("%i", &cuenta->domicilio->CP);
    printf("Ciudad: ");
    fflush(stdin);
    gets(cuenta->domicilio->ciudad);
    printf("Pais: ");
    fflush(stdin);
    gets(cuenta->domicilio->pais);
    break;
```

```
case 3:
    printf("\n ~ Datos nuevo plan ~\n");
    for(i=0; i<6; i++)
        printf("\n\t[%i] %s", i+1, paquetes[i]);
    printf("\nOpcion deseada: ");
    scanf("%i", &tipo);
    while(tipo<1 || tipo>6){
        printf("Opcion no valida, vuelva a ingresar una opcion valida.\nOpcion: ");
        scanf("%i", &tipo);
    }
    strcpy(cuenta->tipoPlan->paquete, paquetes[tipo-1]);
    if (tipo==1 ||tipo==2){
        realloc(cuenta->duenoCuenta, 1*sizeof(Perfil));
        cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
        strcpy(cuenta->tipoPlan->formaPago->formaPago, "NO APLICA");
        strcpy(cuenta->tipoPlan->formaPago->numeroTarjeta, "NO APLICA");
        cuenta->tipoPlan->formaPago->mesVencimiento = 0;
        cuenta->tipoPlan->formaPago->anoVencimiento = 0;
        cuenta->tipoPlan->formaPago->CVV = 0;
        cuenta->tipoPlan->formaPago->precio = 0;
    }else {
        printf("\nTIPO TARJETA");
        for(i=0; i<4; i++)
            printf("\n\t[%i] %s", i+1, tarjetas[i]);
        printf("\nOpcion deseada: ");
        scanf("%i", &i);
        while(i<1 || i>4){
            printf("Opcion no valida, vuelva a ingresar una opcion valida.\nOpcion: ");
            scanf("%i", &i);
        }
```

```
        strcpy(cuenta->tipoPlan->formaPago->formaPago, tarjetas[i-1]);
        printf("\nDATOS TARJETA.\n");
        printf("Numero tarjeta: ");
        fflush(stdin);
        gets(cuenta->tipoPlan->formaPago->numeroTarjeta);
        printf("Mes de vencimiento (nn): ");
        scanf("%i", &cuenta->tipoPlan->formaPago->mesVencimiento);
        printf("ano de vencimiento (nnnn): ");
        scanf("%i", &cuenta->tipoPlan->formaPago->anoVencimiento);
        printf("Codigo de seguridad (CVV): ");
        scanf("%i", &cuenta->tipoPlan->formaPago->CVV);
        switch(tipo){
            case 3:
                cuenta->tipoPlan->formaPago->precio = 99;
                realloc(cuenta->duenoCuenta, 1*sizeof(Perfil));
                cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
                break;

            case 4:
                cuenta->tipoPlan->formaPago->precio = 129;
                realloc(cuenta->duenoCuenta, 2*sizeof(Perfil));
                cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
                break;
            case 5:
                cuenta->tipoPlan->formaPago->precio = 149;
                realloc(cuenta->duenoCuenta, 6*sizeof(Perfil));
                cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
                break;
            case 6:
                cuenta->tipoPlan->formaPago->precio = 49;
                realloc(cuenta->duenoCuenta, 1*sizeof(Perfil));
                cuenta->duenoCuenta[0] = *cuenta->duenoCuenta;
                break;
        }
    }
    break;
```

Esta función se implementa principalmente con casos los cuales cada uno de ellos tiene la funcionalidad de cambiar un dato dentro de la cuenta que van desde la información de perfil de propietario, su dirección y el tipo de paquete que contrató y al mismo tiempo cambia la cantidad de perfiles dentro de la cuenta. Así actualiza el contenido del arreglo cuentas.

d. Buscar

```
int buscarCuenta(Cuenta *cuentas, char correo[], int iCuentas){
    int i;

    for (i=0; i<=iCuentas; i++){
        if(!strcmp(correo, cuentas[i].duenoCuenta->email))
            return cuentas[i].numCuenta;
    }

    return -1;
}

int buscarPerfil(Perfil *perfiles, char correo[], int iPerfiles){
    int i;

    for (i=0; i<=iPerfiles; i++){
        if(!strcmp(correo, perfiles[i].email))
            return perfiles[i].numPerfil;
    }

    return -1;
}
```

Busca recibiendo como parámetro el correo electrónico y checando ya sea contra el arreglo de estructuras Perfil, o el arreglo de estructuras Cuenta para encontrar una coincidencia y devolver el número de perfil o cuenta que coincida. Cada perfil o cuenta tiene una especie de seriación.

e. Listar

```c
void listarCuenta(Cuenta cuenta, Perfil *perfiles){
    int i;
    printf("\n\tCuenta [%i] ...\n", cuenta.duenoCuenta->numCuenta);
    printf("\tPerfil dueno.\n\t\tNombre: %s\n",cuenta.duenoCuenta->nombre);
    printf("\t\tEmail: %s\n",cuenta.duenoCuenta->email);
    printf("\t\tContrasena: %s\n",cuenta.duenoCuenta->contrasena);
    printf("\t\tEdad: %i\n",cuenta.duenoCuenta->edad);
    printf("\t\tDia: %i\n",cuenta.duenoCuenta->dia);
    printf("\t\tMes: %i\n",cuenta.duenoCuenta->mes);
    printf("\t\tAno: %i\n",cuenta.duenoCuenta->ano);
    printf("\t\tGenero: %s\n",cuenta.duenoCuenta->genero);
    printf("\tDomicilio.\n\t\tCalle: %s\n",cuenta.domicilio->calle);
    printf("\t\tNumero: %i\n",cuenta.domicilio->numeroDomicilio);
    printf("\t\tColonia: %s\n",cuenta.domicilio->colonia);
    printf("\t\tCP: %i\n",cuenta.domicilio->CP);
    printf("\t\tCiudad: %s\n",cuenta.domicilio->ciudad);
    printf("\t\tPais: %s\n",cuenta.domicilio->pais);
    printf("\tTipo de plan.\n\t\tPaquete: %s\n",cuenta.tipoPlan->paquete);
    printf("\t\tForma de pago: %s\n",cuenta.tipoPlan->formaPago->formaPago);
    printf("\t\tNumero de tarjeta: %s\n",cuenta.tipoPlan->formaPago->numeroTarjeta);
    printf("\t\tMes de vencimiento: %i\n",cuenta.tipoPlan->formaPago->mesVencimiento);
    printf("\t\tAno de vencimiento: %i\n",cuenta.tipoPlan->formaPago->anoVencimiento);
    printf("\t\tCVV: %i\n",cuenta.tipoPlan->formaPago->CVV);
    printf("\t\tPrecio: %3.2f\n",cuenta.tipoPlan->formaPago->precio);
    printf("\tOtros perfiles.\n");
    if(!strcmp(cuenta.tipoPlan->paquete, "Duo")){
        printf("\t\tperfil 1: %s\n", perfiles[cuenta.perfiles[1]].nombre);
    } else if (!strcmp(cuenta.tipoPlan->paquete, "Familiar")){
        for (i=1; i<5; i++)
            printf("\t\tperfil %i: %s\n", i, perfiles[cuenta.perfiles[i]].nombre);
    } else
        printf("\t\tEn este plan no se tienen perfiles extra.\n");
}

void listarPerfil(Perfil perfil){
    printf("\n\tPerfil [%i]...\n", perfil.numPerfil);
    printf("\t\tNombre: %s\n", perfil.nombre);
    printf("\t\tEmail: %s\n", perfil.email);
    printf("\t\tContrasena: %s\n", perfil.contrasena);
    printf("\t\tEdad: %i\n", perfil.edad);
    printf("\t\tDia de nacimiento: %i\n", perfil.dia);
    printf("\t\tMes de nacimiento: %i\n", perfil.mes);
    printf("\t\tAno de nacimiento: %i\n", perfil.ano);
    printf("\t\tGenero: %s\n", perfil.genero);
    printf("\t\tNumero de cuenta: %i\n", perfil.numCuenta);
}

void listarPlaylist(Playlist playlist){
    int i;

    printf("\n Playlist \"%s\", creador: \"%s\", descargas: %i", playlist.nombrePlaylist, playlist.creador, playlist.descargas);
    for(i=0; i<playlist.numeroCanciones; i++){
        printf("\n\tCancion %i", i+1);
        printf("\n\t\t\"%s\"", playlist.canciones->nombre);
        printf("\n\t\tPor: %s", playlist.canciones->artista);
        printf("\n\t\tDe %i", playlist.canciones->lanzada);
        playlist.canciones++;
    }
    printf("\n\n");
}
```

Cada una de las funciones que se ven es esta foto son para imprimir la información de cada una de las estructuras y arreglos de estructuras que se encuentran registrados.

Se invocan desde el main →

```c
            case 2:
                if (iCuentas == -1)
                    printf("\n No hay cuentas existentes.\n");
                else{
                    for (i=0; i<=iCuentas; i++)
                        listarCuenta(cuentas[i], perfiles);
                }
                break;

            case 3:
                if (iPerfiles == -1)
                    printf("\n No hay perfiles existentes.\n");
                else{
                    for (i=0; i<=iPerfiles; i++)
                        listarPerfil(perfiles[i]);
                }
                break;

            case 4:
                if (iPlaylists == -1)
                    printf("\n No hay playlists existentes.\n");
                else{
                    for (i=0; i<=iPlaylists; i++)
                        listarPlaylist(playlists[i]);
                }
                break;
```

6. Manejo de archivos

```c
int extraerContadores(int *iCuentas, int *iPerfiles, int *iPlaylists);
int guardarContadores(int iCuentas, int iPerfiles, int iPlaylists);

int bajarArchivoCuentas(Cuenta *cuentas, int iCuentas);
int bajarArchivoPerfiles(Perfil *perfiles, int iPerfiles);
int bajarArchivoPlaylists(Playlist *playlists, int iPlaylists);
```

Declaración de las funciones prototipo de los contadores y de los archivos

```c
int extraerContadores(int *iCuentas, int *iPerfiles, int *iPlaylists){
    FILE *archivoContadores;


    archivoContadores = fopen("contadores.txt", "r");
    if(archivoContadores == NULL){
        return 0;
    }
    fscanf(archivoContadores, "%i\n%i\n%i\n", iCuentas, iPerfiles, iPlaylists);
    fclose(archivoContadores);

    return 1;
}

int guardarContadores(int iCuentas, int iPerfiles, int iPlaylists){
    FILE *archivoContadores;

    archivoContadores = fopen("contadores.txt", "w");
    if(archivoContadores == NULL)
        return 0;
    fprintf(archivoContadores, "%i\n%i\n%i\n", iCuentas, iPerfiles, iPlaylists);
    fclose(archivoContadores);

    return 1;
}

int bajarArchivoCuentas(Cuenta *cuentas, int iCuentas){
    int i;
    FILE *archivoCuentas;

    archivoCuentas = fopen("cuentas.txt", "rb");
    if (archivoCuentas == NULL)
        return 0;
```

Se implementaron contadores para poder acceder y leer la información de los archivos creados, así como asignarle nueva información.

```c
int bajarArchivoCuentas(Cuenta *cuentas, int iCuentas){
    int i;
    FILE *archivoCuentas;

    archivoCuentas = fopen("cuentas.txt", "rb");
    if (archivoCuentas == NULL)
        return 0;


    for (i=0; i <= iCuentas; i++){
        crearCuenta(cuentas);
        fscanf(archivoCuentas, "%s %s %s %i %i %i %i %s %i %s %i %s %i %s %s %s %s %s %i %i %i %f %i", cuentas->duenoCuenta->nombre, cuentas->duenoCuenta->en
        if (!strcmp(cuentas->tipoPlan->paquete, "Duo")){
            realloc(cuentas->perfiles, 2*sizeof(int));
            fscanf(archivoCuentas, " %i\n", &cuentas->perfiles[1]);
        } else if (!strcmp(cuentas->tipoPlan->paquete, "Familiar")){
            fscanf(archivoCuentas, " %i %i %i %i\n", &cuentas->perfiles[1], &cuentas->perfiles[2], &cuentas->perfiles[3], &cuentas->perfiles[4]);
        }else{
            realloc(cuentas->perfiles, 1*sizeof(int));
            fscanf(archivoCuentas, "\n");
        }
        descomprimir(cuentas->duenoCuenta->nombre);
        descomprimir(cuentas->duenoCuenta->email);
        descomprimir(cuentas->duenoCuenta->contrasena);
        descomprimir(cuentas->domicilio->calle);
        descomprimir(cuentas->domicilio->ciudad);
        descomprimir(cuentas->domicilio->colonia);
        descomprimir(cuentas->domicilio->pais);
        descomprimir(cuentas->tipoPlan->formaPago->numeroTarjeta);
        cuentas++;
    }
    fclose(archivoCuentas);

    return 1;
}
```

```c
int bajarArchivoPerfiles(Perfil *perfiles, int iPerfiles){
    int i;
    FILE *archivoPerfiles;

    archivoPerfiles = fopen("perfiles.txt", "rb");
    if (archivoPerfiles == NULL)
        return 0;

    for (i=0; i<=iPerfiles; i++){
        fscanf(archivoPerfiles, "%s %s %s %i %i %i %i %s %i %i\n", (perfiles->nombre), (perfiles->email), (perfiles->contrasena), &(perfiles->edad), &(perfiles->dia),
        descomprimir(perfiles->nombre);
        descomprimir(perfiles->email);
        descomprimir(perfiles->contrasena);
        perfiles++;
    }
    fclose(archivoPerfiles);

    return 1;
}

int bajarArchivoPlaylists(Playlist *playlists, int iPlaylists){
    int i, j;
    FILE *archivoPlaylists;

    archivoPlaylists = fopen("playlists.txt", "rb");
    if (archivoPlaylists == NULL)
        return 0;

    for (i = 0; i <= iPlaylists ; i++){
        crearPlaylist(playlists);
        fscanf(archivoPlaylists, "%s %i %s %i", (playlists->nombrePlaylist), &(playlists->numeroCanciones), (playlists->creador), &(playlists->descargas));
        playlists->canciones = crearCanciones(playlists->numeroCanciones);
        for(j = 0; j<playlists->numeroCanciones; j++){
            fscanf(archivoPlaylists, " %s %s %i", playlists->canciones[j].nombre, playlists->canciones[j].artista, &playlists->canciones[j].lanzada);
            descomprimir(playlists->canciones[j].nombre);
            descomprimir(playlists->canciones[j].artista);
        }
        fscanf(archivoPlaylists, "\n");
        descomprimir(playlists->nombrePlaylist);
        descomprimir(playlists->creador);

        playlists++;
    }

    fclose(archivoPlaylists);

    return 1;
}
```

Antes de implementar cualquier función se abren los archivos ya creado y se baja la información existente en ellos con ayuda de los contadores.

```c
int subirArchivoPerfiles(Perfil *perfiles, int iPerfiles){
    int i;
    FILE *archivoPerfiles;

    archivoPerfiles = fopen("perfiles.txt", "wb");
    if (archivoPerfiles == NULL)
        return 0;

    for (i = 0; i <= iPerfiles ; i++){
    comprimir(perfiles->nombre);
    comprimir(perfiles->email);
    comprimir(perfiles->contrasena);
    fprintf(archivoPerfiles, "%s %s %s %i %i %i %i %s %i %i\n", perfiles->nombre, perfiles->email, perfiles->contrasena, perfiles->edad, perfiles->dia, perfiles
        perfiles++;
    }
    fflush(archivoPerfiles);
    fclose(archivoPerfiles);
    return 1;
}

int subirArchivoPlaylists(Playlist *playlists, int iPlaylists){
    int i, j;
    FILE *archivoPlaylists;

    archivoPlaylists = fopen("playlists.txt", "wb");
    if (archivoPlaylists == NULL)
        return 0;


    for (i = 0; i <= iPlaylists ; i++){
        comprimir(playlists->nombrePlaylist);
        comprimir(playlists->creador);
        fprintf(archivoPlaylists, "%s %i %s %i", (playlists->nombrePlaylist), playlists->numeroCanciones, (playlists->creador), playlists->descargas);
        for (j=0; j<playlists->numeroCanciones; j++){
            comprimir(playlists->canciones->nombre);
            comprimir(playlists->canciones->artista);
            fprintf(archivoPlaylists, " %s %s %i", playlists->canciones->nombre, playlists->canciones->artista, playlists->canciones->lanzada);
            playlists->canciones++;
        }
        fprintf(archivoPlaylists, "\n");
        playlists++;
    }
    fflush(archivoPlaylists);
    fclose(archivoPlaylists);
    return 1;
}
```

Se sobrescriben los archivos ya con los cambios realizados durante su ejecución.

7. Implementación de menús

Menu.h

```h
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

int Menu(char texto[], int n);

#endif // MENU_H_INCLUDED

```

Menu.c

```c
#include <stdio.h>
#include "menu.h"

int Menu(char texto[], int n){
    int opcion;
    do {
        printf("%s ", texto);
        scanf("%d", &opcion);
        if (opcion < 1 || opcion > n)
            printf("ERROR: opcion no valida...\n");

    }while (opcion < 1 || opcion > n);

    return opcion;
}

```

A lo largo del proyecto utilizamos varios menúes para dar a elegir al usuario nuestras opciones. En el main de nuestro programa se usaron los siguientes cuatro menúes:

```c
    do{
        opcion = Menu("\n - STREAMING DE MUSICA - \n\t1)Agregar Cuenta\n\t2)Listar Cuentas\n\t3)Listar Perfiles\n\t4)Listar Playlists\n\t5)Acceder
        switch (opcion){
            case 1:
```

El principal para acceder a todas las opciones del usuario: agregar cuenta, listar cuentas, listar perfiles, listar playlists, acceder a una cuenta, acceder a un perfil, generar estadísticas o salir.

```c
            case 5:
                if (checarContrasenaCuenta(&cuentas[0], iCuentas, &numCuenta)){
                    do{
                        opcion = Menu("\n - Menu Cuenta - \n\t1)Mostrar datos\n\t2)Cambiar datos\n\t3)Administrar perfiles\n\t4)Regresar\n Opcion: ", 4);
                        switch (opcion){
                            case 1:
```

Una vez accedes a una cuenta se despliega otro menú para darte a elegir entre las opciones: mostrar los datos, cambiar/modificar los datos, administrar perfiles (añadir o eliminar algún perfil), o regresar.

```
112             case 6:
113                 if (checarContrasenaPerfil(&perfiles[0], iPerfiles, &numPerfil)){
114                     do{
115                         opcion = Menu("\n - Menu Perfil - \n\t1)Mostrar datos\n\t2)Cambiar datos\n\t3)Crear Playlist\n\t4)Regresar\n Opcion: ", 4);
116                         switch (opcion){
117                             case 1:
```

Una vez accedes a un perfil, se despliega otro menú para las opciones mostrar datos, cambiar datos, crear una playlist, o regresar.

```
137             case 7:
138                 do{
139                     opcion = Menu("\n - Estadisticas - \n\t1)Porcentaje de hombres y mujeres\n\t2)Porcentaje de cuentas premium y cuentas free\n\t3)Canti
140                     switch (opcion){
141                     case 1:
```

El último menú del main se encuentra una vez accedes a las estadísticas y te deja elegir cual quieres que te muestre.

En las funciones también contamos con varios menús.

```
563 void cambiarDatosCuenta(Cuenta *cuenta, Perfil *perfiles, int iPerfiles){
564     int i, tipo, opcion, eleccion, aux;
565     char contra_1[30], contra_2[30];
566     char generos[3][12] = {"Masculino", "Femenino", "No-Binario"};
567     char paquetes[6][16] = {"Gratis", "Mes de Prueba", "Individual", "Duo", "Familiar", "Universitario"};
568     char tarjetas[4][18] = {"Visa", "MasterCard", "AmericanExpress", "PayPal"};
569 
570     do{
571         opcion = Menu("\n - Menu Cambio Info Cuenta - \n\t1)Cambiar info dueno de la cuenta \n\t2)Cambiar info domicilio\n\t3)Cambi
572         switch (opcion){
573             case 1:
```

Este menú se despliega cuando quieres cambiar los datos de una cuenta y te dice qué campo de dato quieres cambiar: info del dueño, info del domicilio, o info del plan.

```
737 void cambiarDatosPerfil(Perfil *perfil, Perfil *perfiles, int iPerfiles){
738     int eleccion, aux;
739     char contra_1[30], contra_2[30];
740     char generos[3][12] = {"Masculino", "Femenino", "No-Binario"};
741 
742     listarPerfil(*perfil);
743     do{
744         eleccion = Menu("\n ~ Cambio perfil ~\n\t1)Nombre\n\t2)Email\n\t3)Contrasena\n\t4)Edad\n\t5)Dia de nacimiento\n\t6)Mes\n\t7)Ar
745         switch (eleccion){
746             case 1:
```

Este menú te da a elegir qué dato de un perfil quieres cambiar.

```
834 void gestionarPerfilesCuenta(Cuenta *cuenta, int numCuenta, Perfil *perfiles, int *iPerfiles){
835     int i, eleccion, opcion, aux=0;
836 
837     printf("\n Tu cuenta es...\n");
838     listarCuenta(*cuenta, perfiles);
839     if(!strcmp(cuenta->tipoPlan->paquete, "Duo")){
840         do{
841             eleccion = Menu("\n ~ Menu Gestion Perfiles ~ \n\t1)Agregar perfil\n\t2)Eliminar perfil\n\t3)Regresrar\n Opcion: ", 3);
842             switch (eleccion){
843                 case 1:
```

El último menú te deja añadir o eliminar un perfil.

8. Aritmética de apuntadores

```
42      for (i=0; i <= iCuentas; i++){
43          crearCuenta(cuentas);
44          fscanf(archivoCuentas, "%s %s %s %i %i %i %i %s %i %s %i %s %i %s %s %s %s %s %i %i %i %f %i", cuentas->duenoCuenta->nombre, cuentas->duenoCuenta->email, cuent
45          if (!strcmp(cuentas->tipoPlan->paquete, "Duo")){
46              realloc(cuentas->perfiles, 2*sizeof(int));
47              fscanf(archivoCuentas, " %i\n", &cuentas->perfiles[1]);
48          } else if (!strcmp(cuentas->tipoPlan->paquete, "Familiar")){
49              fscanf(archivoCuentas, " %i %i %i %i\n", &cuentas->perfiles[1], &cuentas->perfiles[2], &cuentas->perfiles[3], &cuentas->perfiles[4]);
50          }else{
51              realloc(cuentas->perfiles, 1*sizeof(int));
52              fscanf(archivoCuentas, "\n");
53          }
54          descomprimir(cuentas->duenoCuenta->nombre);
55          descomprimir(cuentas->duenoCuenta->email);
56          descomprimir(cuentas->duenoCuenta->contrasena);
57          descomprimir(cuentas->domicilio->calle);
58          descomprimir(cuentas->domicilio->ciudad);
59          descomprimir(cuentas->domicilio->colonia);
60          descomprimir(cuentas->domicilio->pais);
61          descomprimir(cuentas->tipoPlan->formaPago->numeroTarjeta);
62          cuentas++;
63      }
```

```
77      for (i=0; i<=iPerfiles; i++){
78          fscanf(archivoPerfiles, "%s %s %s %i %i %i %i %s %i %i\n", (perfiles->nombre), (perfiles->email), (perfiles->contrasena), &(perfiles->edad), &(perfiles->dia), &(perfiles->mes),
79          descomprimir(perfiles->nombre);
80          descomprimir(perfiles->email);
81          descomprimir(perfiles->contrasena);
82          perfiles++;
83      }
```

```
97       for (i = 0; i <= iPlaylists ; i++){
98           crearPlaylist(playlists);
99           fscanf(archivoPlaylists, "%s %i %s %i", (playlists->nombrePlaylist), &(playlists->numeroCanciones), (playlists->creador), &(playlists->descargas));
100          playlists->canciones = crearCanciones(playlists->numeroCanciones);
101          for(j = 0; j<playlists->numeroCanciones; j++){
102              fscanf(archivoPlaylists, " %s %s %i", playlists->canciones[j].nombre, playlists->canciones[j].artista, &playlists->canciones[j].lanzada);
103              descomprimir(playlists->canciones[j].nombre);
104              descomprimir(playlists->canciones[j].artista);
105          }
106          fscanf(archivoPlaylists, "\n");
107          descomprimir(playlists->nombrePlaylist);
108          descomprimir(playlists->creador);
109
110          playlists++;
111      }
```

```
for(i=0; i<playlist.numeroCanciones; i++){
    printf("\n\tCancion %i", i+1);
    printf("\n\t\t\"%s\"", playlist.canciones->nombre);
    printf("\n\t\tPor: %s", playlist.canciones->artista);
    printf("\n\t\tDe %i", playlist.canciones->lanzada);
    playlist.canciones++;
}
```

```
for (i = 0; i <= iCuentas ; i++){
    comprimir(cuentas->duenoCuenta->nombre);
    comprimir(cuentas->duenoCuenta->email);
    comprimir(cuentas->duenoCuenta->contrasena);
    comprimir(cuentas->domicilio->calle);
    comprimir(cuentas->domicilio->ciudad);
    comprimir(cuentas->domicilio->colonia);
    comprimir(cuentas->domicilio->pais);
    comprimir(cuentas->tipoPlan->formaPago->numeroTarjeta);
    if(!strcmp(cuentas->tipoPlan->paquete, "Duo"))
        fprintf(archivoCuentas, "%s %s %s %i %i %i %i %s %i %s %i %s %i %s %s %s %s %s %i %i %i %3.2f %i %i\n", cuentas->duenoCuenta->nombre, cuentas->duenoCuenta->email, cuentas->due
    else if (!strcmp(cuentas->tipoPlan->paquete, "Familiar"))
        fprintf(archivoCuentas, "%s %s %s %i %i %i %i %s %i %s %i %s %i %s %s %s %s %s %i %i %i %3.2f %i %i %i %i %i\n", cuentas->duenoCuenta->nombre, cuentas->duenoCuenta->email, cue
    else
        fprintf(archivoCuentas, "%s %s %s %i %i %i %i %s %i %s %i %s %i %s %s %s %s %s %i %i %i %3.2f %i\n", cuentas->duenoCuenta->nombre, cuentas->duenoCuenta->email, cuentas->duenoC

    cuentas++;
}
```

```
for (i = 0; i <= iPerfiles ; i++){
comprimir(perfiles->nombre);
comprimir(perfiles->email);
comprimir(perfiles->contrasena);
fprintf(archivoPerfiles, "%s %s %s %i %i %i %i %s %i %i\n", perfiles->nombre, perfiles->email, perfiles->contrasena, perfiles->edad, perfiles->dia, perfiles->mes, perfiles->ano, perfi
    perfiles++;
}
fflush(archivoPerfiles);
```

```
for (i = 0; i <= iPlaylists ; i++){
    comprimir(playlists->nombrePlaylist);
    comprimir(playlists->creador);
    fprintf(archivoPlaylists, "%s %i %s %i", (playlists->nombrePlaylist), playlists->numeroCanciones, (playlists->creador), playlists->descargas);
    for (j=0; j<playlists->numeroCanciones; j++){
        comprimir(playlists->canciones->nombre);
        comprimir(playlists->canciones->artista);
        fprintf(archivoPlaylists, " %s %s %i", playlists->canciones->nombre, playlists->canciones->artista, playlists->canciones->lanzada);
        playlists->canciones++;
    }
    fprintf(archivoPlaylists, "\n");
    playlists++;
}
```

La aritmética de apuntadores; se realizó al incrementar los apuntadores correspondientes a cuentas, perfiles y playlists. Con estos incrementos se logró realizar la asignación de información a cada campo de las diferentes estructuras mediante ciclos repetitivos.

10.Arreglo de arreglos de tipos de datos estructuras (struct)
Entendimos el objetivo de este punto cómo agregar un arreglo bidimensional de alguna estructura. Para visualizarlo lo pensamos de la siguiente manera: si quisiéramos registrar estructuras con los datos de una persona, pero queremos dividirlos por profesión, entonces en cada columna del arreglo de personas pondremos las personas con una profesión en específico y en el siguiente las de la otra profesión.
Sin embargo, no le vimos utilidad en nuestro programa ya que no contábamos con una situación parecida a ésta. La manera de implementarlo sería poner datos iguales sin distinción o relación en una matriz en lugar de un vector y eso no era práctico.

9. Seis estadísticas para explotar los datos almacenados y presentar información para toma de decisiones. Deberá de implementar una función por estadística, pasando los datos por referencia.
11 .Agregar 4 estadísticas más, deberán de ser MÍNIMO 10 estadísticas para explotar los datos almacenados y presentar información para toma de decisiones.

Las estadísticas que presentamos en nuestro proyecto son:
Prototipos de funciones:

```
 99   void Estad_1_porcentajeHombresYMujeres(Perfil *perfiles, int iPerfiles);
100   void Estad_2_porcentajeCuentas(Cuenta *cuentas, int iCuentas);
101   void Estad_3_playlistsRegistradas(int iPlaylists);
102   void Estad_4_ingresosRegistrados(Cuenta *cuentas, int iCuentas);
103   void Estad_5_metodoPago(Cuenta *cuentas,int iCuentas);
104   void Estad_6_edadPromedio(Perfil *perfiles,int iPerfiles);
105   void Estad_7_usuariosRegistrados(int iPerfiles);
106   void Estad_8_cuentaMasCostosa(Cuenta *cuentas, int iCuentas);
107   void Estad_9_playlistsReistrados(Playlist *playlists, int iPlaylists);
108   void Estad_10_masCanciones(Playlist *playlists, int iPlaylists);
109   void Estad_11_epoca(Playlist *playlists, int iPlaylists);
```

Las funciones de cada estadística:

```c
void Estad_1_porcentajeHombresYMujeres(Perfil *perfiles, int iPerfiles){
    int i;
    int hombres=0, mujeres=0, otros=0, total=0;
    float porcentajeHombres, porcentajeMujeres;

    for (i=0; i<=iPerfiles; i++){
        if (strcmp(perfiles[i].genero,"Masculino") == 0)
            hombres++;
        else if (strcmp(perfiles[i].genero,"Femenino") == 0)
            mujeres++;
        else
            otros++;
    }
    printf("\n\tHay registros de:\n\t\t%i Hombres\n\t\t%i Mujeres\n\t\t%i No especificado", hombres, mujeres, otros);
    total = hombres + mujeres + otros;
    porcentajeHombres = (hombres*100)/total;
    porcentajeMujeres = (mujeres*100)/total;
    printf("\n\n\t ~~ EL %.2f%% de los usuarios registrados son MUJERES mientras que el %.2f%% son HOMBRES ~~\n\n", porc
}
```

En la estadística uno, se recorren todos los perfiles registrados y se contabiliza si son femeninos o masculinos. Al final se saca el porcentaje de cuánto representan los hombres y mujeres del total de perfiles y se imprime en pantalla este resultado.

```c
void Estad_2_porcentajeCuentas(Cuenta *cuentas, int iCuentas){
    int i;
    int free=0, premium=0, total=0;
    float porcentajeFree, porcentajePremium;

    for (i=0; i<=iCuentas; i++){
        if (strcmp(cuentas[i].tipoPlan->paquete,"Gratis") == 0)
            free++;
        else if (strcmp(cuentas[i].tipoPlan->paquete, "Mes-Prueba") == 0)
            free++;
        else
            premium++;
    }
    printf("\n\tHay registros de:\n\t\t%i Cuentas Gratis\n\t\t%i Cuentas Premium", free, premium);
    total = free + premium;
    porcentajeFree = (free*100)/total;
    porcentajePremium = (premium*100)/total;
    printf("\n\n\t ~~ El %.2f%% de los usuarios registrados tienen CUENTAS GRATIS mientras que el %.2f%% tienen CUENTAS PREMIUM ~~\n\n", pc
}
```

En la estadística dos, se recorren todas las cuentas registradas y se va contabilizando si son cuentas gratis o cuentas Premium (de paga). Al final se imprime en pantalla la relación de porcentajes de cada una. Para contar las cuentas gratis se checa si su paquete es “Gratis” o “Mes-Prueba”, los demás son de paga.

```c
void Estad_3_playlistsRegistradas(int iPlaylists){
    printf("\n\n\t ~~ Hay %i playlists registradas en la plataforma. ~~\n\n", iPlaylists+1);
}
```

En la tercera estadística se usa el contador de las playlists registradas para imprimir en pantalla este número. Se le suma uno al contador de Playlists, ya que este empieza en -1, entonces la primera es 0 y así sucesivamente.

```c
void Estad_4_ingresosRegistrados(Cuenta *cuentas, int iCuentas){
    int i;
    float totalIngreso=0;

    for(i=0; i<=iCuentas; i++)
        totalIngreso+= cuentas[i].tipoPlan->formaPago->precio;
    printf("\n\t ~~ El ingreso mensual de la plataforma asciende a $%.2f MXN ~~\n\n", totalIngreso);

}
```

En la cuarta estadística se contabiliza las ganancias de la plataforma al recorrer el arreglo de la estructura cuenta e ir sumando el precio de los paquetes de cada cuenta. Al final se imprime en pantalla las ganancias mensuales de la plataforma.

```c
void Estad_5_metodoPago(Cuenta *cuentas,int iCuentas){
    int i, Visa = 0, MasterCard = 0, AmericanExpress = 0, Paypal = 0;
    printf("\n\n\t El metodo mas usual de pago:");
    for(i=0; i<=iCuentas;i++){
        if(!strcmp("Visa",cuentas[i].tipoPlan->formaPago->formaPago))
            Visa++;
        else if(!strcmp("MasterCard",cuentas[i].tipoPlan->formaPago->formaPago))
            MasterCard++;
        else if(!strcmp("AmericanExpress",cuentas[i].tipoPlan->formaPago->formaPago))
            AmericanExpress++;
        else if(!strcmp("PayPal",cuentas[i].tipoPlan->formaPago->formaPago))
            Paypal++;
    }
    if(Visa>=MasterCard && Visa>=AmericanExpress && Visa>=Paypal)
        printf("\n\t ~~ Visa ~~");
    if(MasterCard>=Visa && MasterCard>=AmericanExpress && MasterCard>=Paypal)
        printf("\n\t ~~ MasterCard ~~");
    if(AmericanExpress>=Visa && AmericanExpress>=MasterCard && AmericanExpress>=Paypal)
        printf("\n\t ~~ AmericanExpress ~~");
    if(Paypal>=Visa && Paypal>=MasterCard && Paypal>=AmericanExpress)
        printf("\n\t ~~ Paypal ~~");
    printf("\n\n");
}
```

La quinta estadística recorre nuevamente el arreglo de cuentas y contabiliza en cuatro variables cuántas veces se ha pagado con cada método de pago. Esto al comparar la cadena de caracteres de un campo de la estructura cuentas con las opciones de pago. Al final imprime en pantalla el método más usado, si hay un empate, imprime los que empataron.

```
1010  void Estad_6_edadPromedio(Perfil *perfiles,int iPerfiles){
1011      int i;
1012      float edadprom = 0;
1013      for(i=0; i<=iPerfiles; i++){
1014          edadprom = edadprom + perfiles[i].edad;
1015      }
1016      edadprom = edadprom/(iPerfiles+1);
1017      printf("\n\n\t ~~ Edad promedio de los usuarios que usan Spotify: %2.1f ~~\n\n", edadprom);
1018  }
1019
1020  void Estad_7_usuariosRegistrados(int iPerfiles){
1021      printf("\n\n\t ~~ El numero de usuarios registrados es: %i ~~\n\n",iPerfiles+1);
1022  }
```

La sexta estadística recorre el arreglo de perfiles y saca el promedio de la edad de todos los perfiles. Al final imprime en pantalla la edad promedio de los usuarios.

```
1020  void Estad_7_usuariosRegistrados(int iPerfiles){
1021      printf("\n\n\t ~~ El numero de usuarios registrados es: %i ~~\n\n",iPerfiles+1);
1022  }
```

La séptima estadística usa el contador de perfiles para imprimir en pantalla cuantos perfiles registrados hay. Se le suma uno al contador, ya que este se inicializa en -1.

```
1024  void Estad_8_cuentaMasCostosa(Cuenta *cuentas, int iCuentas){
1025      int i;
1026      float mayorCosto=0;
1027
1028      for (i=0; i<iCuentas; i++)
1029          if(cuentas[i].tipoPlan->formaPago->precio>mayorCosto)
1030              mayorCosto = cuentas[i].tipoPlan->formaPago->precio;
1031
1032      printf("\n\tLa cuenta mas costosa registrada es:\n~~\t\t%s\t\t%s\t\t%.2f ~~\n\n", cuentas[i].duenoCuent
1033  }
```

La estadística ocho recorre las cuentas e imprime en pantalla los datos de la cuenta que más paga. Esto lo hace al comparar si la siguiente cuenta, paga más que la anterior.

```
1035  void Estad_9_playlistsReistrados(Playlist *playlists, int iPlaylists){
1036      int i, x=0, xp;
1037
1038      printf("\n\tTotal de playlists: %i", iPlaylists+1);
1039
1040      for(i=0; i<=iPlaylists; i++)
1041          x = x + playlists[i].numeroCanciones;
1042      xp = x/(iPlaylists+1);
1043      printf("\n\t ~~ El numero total de canciones es de: %i ~~", x);
1044      printf("\n\t ~~ El promedio de canciones por playlist es: %i ~~\n\n", xp);
1045
1046  }
```

La estadística nueve da información sobre los playlists registrados. Recorre el archivo de playlists, saca el total de canciones de la plataforma y el promedio de canciones de cada playlist. Esto lo muestra en pantalla.

```
1048 void Estad_10_masCanciones(Playlist *playlists, int iPlaylists){
1049     int i, mayorPlay = 0, aux;
1050     for (i=0; i<=iPlaylists; i++){
1051         if(playlists[i].numeroCanciones > mayorPlay){
1052             mayorPlay = playlists[i].numeroCanciones;
1053             aux = i;
1054         }
1055     }
1056     printf("\n\t ~~ La playist con mas canciones es \"%s\" del perfil %s y tiene:  %i canciones ~~\n\n", playlists[aux].nomb
1057 }
```

La estadística 10 checa cuál playlist tiene más canciones e imprime su nombre, su creador y número de canciones en pantalla. Para hacer esto recorre el arreglo de playlists y si encuentra una con más canciones guarda la posición de ésta en lugar de la anterior.

```
1059 void Estad_11_epoca(Playlist *playlists, int iPlaylist){
1060     int i, j, total=0, totCanciones=0;
1061     for (i=0; i<=iPlaylist; i++){
1062         for (j=0; j<playlists[i].numeroCanciones; j++){
1063             total += playlists[i].canciones[j].lanzada;
1064             totCanciones++;
1065         }
1066     }
1067     printf("\n\t ~~ En promedio las canciones que se escuchan se lanzaron en: %i ~~\n\n", total/totCanciones);
1068 }
```

La estadística 11 calcula el promedio del año en que fueron lanzadas todas las canciones de la plataforma, así se sabe de qué época les gusta escuchar más música. Esto lo hace recorriendo el arreglo de playlists y el arreglo dentro de playlists de canciones.

Aquí se muestra la invocación de las estadísticas desde el main.

```
138                 do{
139                     opcion = Menu("\n - Estadisticas - \n\t1)Porcentaje de hombres y mujeres\n\t2)
140                     switch (opcion){
141                     case 1:
142                         if (iPerfiles>-1)
143                             Estad_1_porcentajeHombresYMujeres(perfiles,iPerfiles);
144                         else
145                             printf("\n No hay datos registrados...\n");
146                         break;
147                     case 2:
148                         if (iCuentas>-1)
149                             Estad_2_porcentajeCuentas(cuentas,iCuentas);
150                         else
151                             printf("\n No hay datos registrados...\n");
152                         break;
153                     case 3:
154                         if (iPlaylists>-1)
155                             Estad_3_playlistsRegistradas(iPlaylists);
156                         else
157                             printf("\n No hay datos registrados...\n");
158                         break;
159                     case 4:
160                         if (iCuentas>-1)
161                             Estad_4_ingresosRegistrados(cuentas,iCuentas);
162                         else
163                             printf("\n No hay datos registrados...\n");
164                         break;
165                     case 5:
166                         if (iCuentas>-1)
167                             Estad_5_metodoPago(cuentas, iCuentas);
168                         else
169                             printf("\n No hay datos registrados...\n");
170                         break;
```

```
                case 6:
                    if (iPerfiles>-1)
                        Estad_6_edadPromedio(perfiles, iPerfiles);
                    else
                        printf("\n No hay datos registrados...\n");
                    break;
                case 7:
                    if (iPerfiles>-1)
                        Estad_7_usuariosRegistrados(iPerfiles);
                    else
                        printf("\n No hay datos registrados...\n");
                    break;
                case 8:
                    if (iCuentas>-1)
                        Estad_8_cuentaMasCostosa(cuentas,iCuentas);
                    else
                        printf("\n No hay datos registrados...\n");
                    break;
                case 9:
                    if (iPlaylists>-1)
                        Estad_9_playlistsReistrados(playlists, iPlaylists);
                    else
                        printf("\n No hay datos registrados...\n");
                    break;
                case 10:
                    if (iPlaylists>-1)
                        Estad_10_masCanciones(playlists, iPlaylists);
                    else
                        printf("\n No hay datos registrados...\n");
                    break;
                case 11:
                    if (iPlaylists>-1)
                        Estad_11_epoca(playlists, iPlaylists);
                    else
                        printf("\n No hay datos registrados...\n");
                    break;
                }
            }while(opcion != 12);
            break;
        }
```

12. Estructuras compuestas por arreglos de estructuras

Dentro del programa contamos con un arreglo de estructuras como variable de otra estructura en el siguiente ejemplo:

Estructuras:

```
typedef struct{
    char *nombre;
    char *artista;
    int lanzada;
}Cancion;

typedef struct{
    char *nombrePlaylist;
    int numeroCanciones;
    char *creador;
    int descargas;
    Cancion *canciones;
} Playlist;
```

```c
void capturarPlaylist(Playlist *playlist, char creador[]){
    int i;

    crearPlaylist(playlist);
    printf("\n Nombre playlist: ");
    fflush(stdin);
    gets(playlist->nombrePlaylist);
    printf(" Numero de canciones: ");
    scanf("%i", &playlist->numeroCanciones);
    playlist->canciones = crearCanciones(playlist->numeroCanciones);
```

```c
Cancion *crearCanciones(int numCanciones){
    int i;
    Cancion *canciones;

    canciones = (Cancion *) calloc(numCanciones, sizeof(Cancion));
    if (canciones == NULL){
        printf("Error al crear canciones");
        exit(-1);
    }
    for (i=0; i<numCanciones; i++){
        canciones[i].nombre = (char *) calloc(30, sizeof(char));
        if (canciones[i].nombre == NULL){
            printf("Error al crear nombre cancion");
            exit(-1);
        }
        canciones[i].artista = (char *) calloc(30, sizeof(char));
        if (canciones[i].artista == NULL){
            printf("Error al crear nombre artista");
            exit(-1);
        }
    }
    return canciones;
}
```

La estructura Playlist contiene en ella un apuntador hacia un arreglo de estructuras Cancion, el cual se inicializa al capturar una playlist, y su número de elementos dependerá del número de canciones previamente especificado en numeroCanciones.

Conclusiones y reflexiones individuales:

Gómez Martínez Cristopher Emiliano

El proyecto que realizamos mis compañeros y yo fue a mi parecer de los complicados, y es que debimos de usar todo el conocimiento que adquirimos durante el semestre pasado así como lo que hemos visto en el transcurso de este semestre y aunque fue algo complicado trabajar en equipo debido a que no podemos hablar en persona en algunos casos nos costó algo de trabajo estar al 100% de acuerdo conforme a cómo debíamos hacer el proyecto , pero logramos superar estos contratiempos y es gracias a todas las prácticas que hemos realizado con anterioridad ya que lo pedido en el proyecto, es todo lo que hemos realizado dentro de esas práctica, solo que con más de nivel de complejidad debido al número de estructuras realizadas, las funciones que se debieron de implementar, el manejo de los archivos para que todos los datos se guardarán y que también se modificarán en el “correr” del programa.

Lugo Sáenz Jesús

Con este proyecto, pudimos superar varios retos, ya que por el motivo de esta contingencia no nos fue tan fácil comunicarnos, pero al final quedó un trabajo que a mi parecer es muy bueno y que gracias con el trabajo en equipo pudimos realizarlo y acabarlo en el tiempo indicado, a mí al inicio de este curso, me costaba trabajo programar, pero poco a poco fue adquiriendo un poquito más de nivel, y con este proyecto quedó reafirmado ese conocimiento. Y aunque nuestra planeación de cómo sería el proyecto se escuchaba complicada, pudimos terminarla de manera correcta y sin caer en pánico. Me siento muy satisfecho al poder lograr junto con mi equipo un proyecto que a mi parecer es muy interesante y que todos estos conocimientos que adquirimos, los sigamos utilizando para casos que se presenten en la vida cotidiana.

Nolasco Sotelo Brenda Carolina

El proyecto presentado anteriormente fue elaborado en conjunto con tres de mis compañeros de clase. Debido a la situación por la que estamos pasando no trabajamos como lo habíamos planeado sino que nos vimos en la necesidad de seccionar las actividades para poder elaborarlas, pues no pudimos reunirnos presencialmente. Pese a esto considero que realizamos un buen proyecto y que nos organizamos de la mejor manera posible porque nos propusimos realizar también el nivel avanzado y no solo el nivel básico.

En cuanto a los aprendizajes que adquirí, muchas de las dudas que me quedaron de algunas prácticas pasadas se aclararon y claro también surgieron nuevas, pero que espero ir resolviendo conforme siga programando. Pude darme cuenta que hacer código eficiente es muy importante para no hacerlo tan extenso pero también para poder volver a utilizarlo y hacerlo aplicativo para situaciones reales. El empleo de funciones me quedó más claro así como también el paso de parámetros por valor y por referencia que anteriormente me confundían mucho. Sin lugar a dudas este proyecto me ayudó bastante para mejorar bases de programación que no tenía tan claras.

Ordiales Caballero Iñaky

Este fue el primer proyecto de un tamaño considerable que realicé en equipo. En él me pude dar cuenta de lo complicado que puede ser ponerse de acuerdo para hacer las partes por separado y luego juntarlas. Siento que a veces no nos entendíamos del todo, sin embargo nos pudimos sobreponer a esto, mejorar nuestra comunicación y acabar creando un programa que a mí me deja bastante satisfecho. Aprendí la importancia de hacer código general por dos razones: una poder re-utilizarlo en otra parte del programa y dos para poder juntar las partes trabajadas en equipo y ajustarlas adecuadamente. Además de lo antes mencionado, la elaboración del código sí fue un reto, ya que se trataba de utilizar todas las habilidades con las que contábamos para programar, además de otras que tuvimos que adquirir sobre la marcha (manejo de archivos). Sin embargo se siente satisfactorio el utilizarlo todo en un programa que compila y ejecuta sin errores. Finalmente el mencionar que este proyecto se podría usar como base para un mayor desarrollo en el futuro.