```
// Lopez Michel Jose Alonso   Matricula: 370650
// Fecha inicio: 11/10/2023   Fecha fin: 17/10/2023
// En esta libreria por medio de los struct podemos hacer datos para hacer registros y en base a las funcione
// poder usar los datos de las variables del struct, para en base a eso poder hacer registros para poder
// agregar,ordenar,imprimir,buscar y ordenar dichos datos en la ejecucion del programa.
// JoseLib.h

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h> //Declaramos las librerias que usaremos en la libreria.

//         ./semana10/LMJA_A10_01_432
//*************************************************************************************************************************
typedef struct Tkey
{
    long matricula;
    char nombre[31];
    char apellido_paterno[31];
    char apellido_materno[31];
    int estatus;
    int genero;
    int edad;

}Tkey;      //Aqui declaramos las variables de la estructura de los datos que pondra en un futuro el usuario.
//*************************************************************************************************************************


//*************************************************************************************************************************
int validaL(long ri, long rf,const char msg[],const char msgError[])
{
   //Son las condiciones para los valores de tipo caracter para validar que lo sean.

    long num;
    char xnum[30];  //Declaramos las variables y la cadena.

    do
    {
        puts(msg);    //Imprimimos el mensaje de opcion de los datos de los campos que usaremos en la estructura del programa.
        fflush(stdin); //Limpiamos el buffer de la basura que se puede llegar a almacenar.
        gets(xnum);     //Le pedimos al usuario que nos de el valor para la cadena.
        num = atoi(xnum);  //convertimos el valor de la cadena xnum a un valor entero num.

        if (num < ri || num > rf)  //Si el valor entero del numero es menor a ri o si es mayor a rf, imprimir el mensaje de error que dice que
no se escogio una de las opciones predispuestas.
        {
            printf("%s", msgError); //Imprimimos el mensaje de error.
            printf("\n");
        }
    } while (num < ri || num > rf); //Y el programa seguira mientras numero sea menor a ri y mayor a rf.

    return num; //Retornamos el valor numero ya que lo ocuparemos.
}
//*************************************************************************************************************************


//*************************************************************************************************************************
int validaInt(int ri, int rf,const char msg[],const char msgError[])
{
    //Son las condiciones para los valores de tipo entero para validar que lo sean.

    int num;
    char xnum[30]; //Declaramos las variables y la cadena.

    do
    {
        puts(msg);     //Imprimimos el mensaje de opcion de los datos de los campos que usaremos en la estructura del programa.
        fflush(stdin);  //Limpiamos el buffer de la basura que se puede llegar a almacenar.
        gets(xnum);      //Le pedimos al usuario que nos de el valor para la cadena.
        num = atoi(xnum);  //convertimos el valor de la cadena xnum a un valor entero num.

        if (num < ri || num > rf) //Si el valor entero del numero es menor a ri o si es mayor a rf, imprimir el mensaje de error que dice que
no se escogio una de las opciones predispuestas.
        {
            printf("%s", msgError); //Imprimimos el mensaje de error.
            printf("\n");
        }
    } while (num < ri || num > rf); //Y el programa seguira mientras numero sea menor a ri y mayor a rf.

    return num; //Retornamos el valor numero ya que lo ocuparemos.
}
//*************************************************************************************************************************


//*************************************************************************************************************************
int busqSec(Tkey *registros, int tam, long matricula)
```

```
{
  //Comienza desde el primer elemento de la lista, compara el elemento actual con el valor que estás buscando.Si el elemento
  //actual coincide con el valor que estás buscando, has encontrado el elemento y puedes detener la búsqueda.
  // Si no coincide, pasa al siguiente elemento en la lista y compara el elemento actual con el valor que estás buscando.
  // Si se ha revisado todos los elementos de la lista y no se encontro el valor, entonces el valor no está en la lista.

    int i;
    for (i = 0; i < tam; i++)  //Dictamos que i comienze en 0, siga mientras 0 sea menor que 100 y que vaya en uno en uno.
    {
        if(registros[i].matricula == matricula) //Buscamos que si los registros de la matricula son iguales a el campo llave de la matricula
        {
            return i;   //retornara el valor que corresponde a i.
        }
    }

    return -1;
}
//*********************************************************************************************************************************


//*********************************************************************************************************************************
void burbuja(Tkey vect[], int tam)
{
    // Comienza comparando el primer elemento de la lista con el segundo. Si el primer elemento es mayor que el segundo, se intercambian.
    // Luego, compara el segundo elemento con el tercero y repite el intercambio si es necesario.
    // Continúa este proceso de comparar y, si es necesario, intercambiar, avanzando a través de la lista hasta el penúltimo elemento.
    // Realiza este proceso  hasta que no haya más elementos para comparar.
    // Cada iteración se conoce como una pasada. En cada pasada, el elemento más grande burbujea hacia la posición correcta,
    // al final de la lista. Después de una pasada completa, reinicia el proceso desde el principio (excluyendo el último elemento que
    // ya está en su posición final, ya que es el más grande).
    // Repite las pasadas hasta que no se realicen intercambios en una pasada completa, lo que indica que la lista está ordenada.


    system("CLS");
    int i, j;
    Tkey burbuja;
    for(i = 0; i < tam - 1; i++)
    {
        for(j = i + 1; j < tam; j++)
        {
            if (vect[j].matricula < vect[i].matricula)
            {
                burbuja = vect[j];
                vect[j] = vect[i];
                vect[i] = burbuja;
            }
        }
    }

    system("PAUSE");
}
//*********************************************************************************************************************************


//*********************************************************************************************************************************
void quickSort(Tkey registros[], int limIzq, int limDer)
{
    int izq, der, tem;
    Tkey central;    //Declaramos las variables.

    izq = limIzq;
    der = limDer;  //Dos índices limIzq y limDer que representan los límites izquierdo
                   //y derecho del segmento del arreglo a ordenar.

    central.matricula = registros[(izq + der) / 2].matricula;
                                  //calcula el elemento central en el segmento de la lista que se está ordenando.

    do{
        //Utilizamos un bucle do-while para realizar el proceso de particionamiento y ordenamiento dentro del
        //segmento definido por los límites limIzq y limDer.

        //Utilizamos dos bucles while para encontrar elementos en el lado izquierdo y derecho del segmento que deben ser intercambiados.
        while(registros[izq].matricula < central.matricula && izq < limDer)
        {

            izq++;
        }

        while(central.matricula < registros[der].matricula && der > limIzq)
        {
            der--;
        }

        //Realizamos los intercambios necesarios si encuentra elementos que deben ser reorganizados.
        if ( izq <= der)
        {
            //Realizamos un intercambio (swap) de elementos en el arreglo registros, específicamente
```

```
            //intercambiando los elementos en las posiciones izq y der si se cumple la condición de que izq es menor o igual a der.
            tem = registros[izq].matricula;
            registros[izq].matricula = registros[der].matricula;
            registros[der].matricula = tem;
            izq++;
            der--;
        }

    }while(izq <= der);

    if (limIzq < der)
    {
        quickSort(registros, limIzq, der);
    }

    if (limDer > izq)
    {
        quickSort(registros, izq, limDer);
    }
}
//*************************************************************************************************************************



//*************************************************************************************************************************
//ordenar un arreglo de tipo Talum usando dos posibles algoritmos de ordenamiento: el algoritmo de la burbuja (burbuja)
// o el algoritmo Quick Sort (quickSort), dependiendo de la bandera band.

int ordenarR(Tkey registros[], int tam, int band)
{
    system("cls");
    //verifica si el tamaño tam del arreglo es mayor que 0. Si es 0 o menor, imprime un mensaje de error y devuelve 0.
    if(tam > 0)
    {
        //Si band es 0, usa el algoritmo de la burbuja para ordenar el arreglo llamando a la función burbuja. Luego, cambia el valor de band a
1.
        if(band == 0)
        {
            burbuja(registros, tam);
            band = 1;
        }
        else //Si band no es 0, utiliza el algoritmo Quick Sort para ordenar el arreglo llamando a la función quickSort.
        {
            quickSort(registros, 0, tam);
        }

        printf("HAS ORDENADO LOS DATOS\n");
        system("PAUSE");
        return band;
    }
    else
    {
        printf("PRIMERO DEBE HABER DATOS\n");
        system("PAUSE");
        return 0;
    }
}
//*************************************************************************************************************************



//*************************************************************************************************************************
int eliminarEsp(char cadena[])
{
    int tam; // Variable para almacenar el tamaño de la cadena
    int i;   // Variable para iterar a través de la cadena

    tam = strlen(cadena);  //Se utiliza la función strlen para obtener el tamaño de la cadena cadena.

    if(cadena[0] == '\0')  //Si la cadena está vacía (es decir, el primer carácter es el carácter nulo '\0'), se retorna 0.
    {
        return 0;
    }

    for(i = 0; i <= tam; i++) //Se utiliza un bucle for para iterar a través de la cadena.Dentro del bucle, se verifica
    {
        if(cadena[i] == ' ' && cadena[i + 1] == ' ') //si hay espacios consecutivos (' ') en la cadena. Si se encuentran
                                                     //espacios consecutivos, la función retorna -1.
        {
            return -1;
        }
    }
    return 0;
}
//*************************************************************************************************************************



//*************************************************************************************************************************
int soloLetras(char cadena[])
```

```
{
    int tam;
    int i;

    tam = strlen(cadena); //Se utiliza la función strlen para obtener el tamaño de la cadena cadena.

    if(eliminarEsp(cadena) == 0) //Llama a la función eliminarEsp(cadena) y verifica si retorna 0.
                                 //Si retorna 0, indica que la cadena no tiene espacios consecutivos y procede a la siguiente verificación. Si
no, retorna 0.
    {
        return 0;
    }

    for(i =  0; i <= tam; i++)
    {
        //Se utiliza un bucle for para iterar a través de la cadena.
        //Dentro del bucle, verifica si cada carácter de la cadena no es una letra. Si no es una letra, retorna -1.
        if(cadena[i] < 'A' || cadena[i] > 'z')
        {
            return -1;
        }
        if(cadena[i] > 'Z' && cadena[i] < 'a')
        {
            return -1;
        }
        //Se verifica tanto mayúsculas como minúsculas, asegurándose de que estén dentro del rango de letras en el código ASCII.
        //Se devuelve 0 si todos los caracteres son letras.
        return 0;
    }
    return 0;
}
//*********************************************************************************************************************


//*********************************************************************************************************************
void validaCad(char cadena[], const char *msg)
{
    int tam, band; //Se declara la función validaCad que toma un arreglo de caracteres cadena[] y un puntero a constante msg.
                   //Se declaran dos variables enteras tam y band.

    band = 1;  //Inicializar la variable band con el valor 1.
    do{
        //Utiliza un bucle do-while que se ejecuta al menos una vez y continúa ejecutándose mientras band tenga el valor 1.

        printf("%s", msg); //Imprime el mensaje proporcionado por el puntero msg.
        fflush(stdin); //Limpia el buffer de entrada usando fflush(stdin).
        gets(cadena); //Lee una línea de texto desde la entrada estándar y la almacena en cadena.

        if (soloLetras(cadena) == 0)
        {
            if(eliminarEsp(cadena) == -1)
            {
                printf("NO SE PERMITEN ESPACIOS\n");
                continue;
            }
        }
        else
        {
            if(cadena[0] == ' ')
            {
                printf("NO SE PERMITEN ESPACIOS\n");
                continue;
            }
            else
            {
                printf("INGRESA SOLO LETRAS\n");
                continue;
            }
        }
        //Llama a la función soloLetras para verificar si la cadena contiene solo letras.Si contiene solo letras, verifica si hay espacios
        //consecutivos utilizando eliminarEsp.Imprime mensajes de error apropiados según la validación realizada y continua el bucle si se
detectan errores.

        strupr(cadena); //Convierte la cadena a mayúsculas utilizando strupr.
        band = 0; //Cambia el valor de band a 0, lo que indica que no se deben realizar más iteraciones en el bucle.

    }while(band == 1);
}
//*********************************************************************************************************************


//*********************************************************************************************************************
int existeTkey(long mat, Tkey *reg, int tam)
{
    int i;

    for(i = 0; i <= tam; i++) //Utiliza un bucle for para iterar a través del arreglo de estructuras Talum.
```

```
    {
        if (mat == reg[i].matricula) //Dentro del bucle, verifica si la matrícula (mat) coincide con la matrícula de la estructura actual del
arreglo.
        {
            return 1; //Si encuentra una coincidencia, retorna 1 indicando que la matrícula existe en el arreglo.
        }
    }
    return 0;
}
//*************************************************************************************************************************


//*************************************************************************************************************************
Tkey datosR(Tkey registros[], int tam)
{
    Tkey reg;
    long matricula;
    char *nombre;
    char *apellido;
    int estatus;
    int sexo;
    int edad;
    //Se declaran varias variables para almacenar información como matrícula, nombre, apellido, estatus, sexo y edad.
    //Se declara una estructura Talum llamada reg para almacenar los datos de un estudiante.

    matricula = (rand() % 99999) + 300000;
    while(existeTkey(matricula, registros, tam) == 1)
    {
        matricula = (rand() % 99999) + 300000;
    }   //Genera una matrícula aleatoria en el rango entre 300,000 y 399,999.
        //Utiliza un bucle para asegurarse de que la matrícula generada sea única en el arreglo registros usando la función existeTkey.

    edad = (rand() % 60) + 17;
    estatus = rand() % 2;
    sexo = rand() % 2;   //Genera datos aleatorios para la edad, estatus y sexo del estudiante.

    reg.estatus = estatus;
    reg.matricula = matricula;
    reg.edad = edad;
    reg.genero = sexo;       //Asigna los datos generados a la estructura reg correspondiente.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    char nombresM[][31] = {"Juan", "Luis", "Carlos", "Pedro"};
    char nombresF[][31] = {"Maria", "Ana", "Laura", "Sofia"};
    char apellidosP[][31] = {"Garcia", "Rodriguez", "Martínez", "Lopez", "Perez", "Fernandez", "Gonzalez", "Hernandez"};
    char apellidosM[][31] = {"Sanchez", "Ramirez", "Torres", "Diaz", "Vargas", "Jimenez", "Ruiz", "Silva"};
    int apellido_paternoIndex = rand() % 8;
    int apellido_maternoIndex = rand() % 8;
    if (sexo == 1)
    {
        // Genera un índice aleatorio para nombres masculinos
        int nombreIndex = rand() % 4;
        strcpy(reg.nombre, nombresM[nombreIndex]);
    }
    else
    {
        // Genera un índice aleatorio para nombres femeninos
        int nombreIndex = rand() % 4;
        strcpy(reg.nombre, nombresF[nombreIndex]);
    }

    /*strcpy(reg.nombre, nombres[nombreIndex]);*/
    strcpy(reg.apellido_paterno, apellidosP[apellido_paternoIndex]);
    strcpy(reg.apellido_materno, apellidosM[apellido_maternoIndex]);

    //Se tienen listas de nombres y apellidos masculinos y femeninos predefinidos.
    //Se elige aleatoriamente un nombre y dos apellidos, dependiendo del sexo.
    //Se asignan estos nombres y apellidos a la estructura reg.

    return reg;
}
//*************************************************************************************************************************


//*************************************************************************************************************************
int agregarR(Tkey registros[], int tam)
{
    system("CLS");
    int i;
    int band = 0; //Se declaran variables enteras i e band para iterar y controlar el proceso de generación de registros.

    for(i = 0; i < 100; i++) //Utiliza un bucle for para generar 100 registros.
    {
        system("CLS");
        registros[tam] = datosR(registros, tam); //En cada iteración, se generan datos aleatorios para un estudiante
                                                 //utilizando la función datosR y se asignan al arreglo de registros en la posición tam.
        system("CLS");
```

```
        tam++; //Después de cada iteración, se incrementa tam para mantener un registro del tamaño actual del arreglo de registros.
    }

    printf("HAS GENERADO 100 REGISTROS\n");
    system("PAUSE");

    return tam;
}
//*****************************************************************************************************************


//*****************************************************************************************************************
Tkey datosM(Tkey registros[], int tam)
{

    Tkey reg; //Se declara una estructura Talum llamada reg para almacenar los datos de un estudiante.
    long mat; //Se declara la función datosM que toma un arreglo de estructuras Talum llamado registros y un entero tam.
    char apPat[21], apMat[21], nombre[41]; //Se declaran variables para almacenar la matrícula, apellidos y nombre del estudiante.

    do{
        //Utiliza un bucle do-while para solicitar y validar los datos del estudiante.

        reg.estatus = validaInt(0, 1,"0) INACTIVO 1) ACTIVO","FUERA DE RANGO");
        mat = validaL(300000, 399999,"INGRESA TU MATRICULA: ","MATRICULA INVALIDA");
        //Se solicita el estatus, matrícula, apellidos, nombre, edad y género del estudiante.

        if(existeTkey(mat, registros, tam) == 1) //Se utiliza la función existeTkey para verificar si la matrícula ingresada ya existe en los
registros.
        {

            printf("ESTA MATRICULA YA EXISTE \n INGRESE UNA MATRICULA VALIDA\n");

        }

    }while(existeTkey(mat, registros, tam) == 1);

    reg.matricula = mat;

    validaCad(apPat, "INGRESA TU APELLIDO PATERNO: \n");
    strcpy(reg.apellido_paterno, apPat);

    validaCad(apMat, "INGRESA TU APELLIDO MATERNO: \n");
    strcpy(reg.apellido_materno, apMat);

    validaCad(nombre, "INGRESA TU nombre: \n");
    strcpy(reg.nombre, nombre);

    reg.edad = validaInt(17, 80,"INGRESA LA EDAD: ","EDAD FUERA DE RANGO");
    reg.genero = validaInt(1, 2,"INGRESE EL GENERO\n 1) HOMBRE\n 2) MUJER","OPCION FUERA DE RANGO");

    //Asigna los datos ingresados por el usuario a la estructura reg correspondiente.

    return reg;
}
//*****************************************************************************************************************


//*****************************************************************************************************************
int agregarM(Tkey registros[], int tam)
{
    int op = 1;

    do{
        //Se utiliza un bucle do-while para permitir al usuario agregar múltiples registros o salir del proceso.

        system("CLS");
        registros[tam] = datosM(registros, tam); //Genera datos de un estudiante utilizando la función datosM y
                                                 //asigna estos datos al arreglo de registros en la posición tam.
        system("CLS");
        tam++;  //Incrementa tam para mantener un registro del tamaño actual del arreglo de registros.

        op = validaInt(0, 1,"1.- AGREGAR DATOS \n0.- SALIR \n SELECCIONE UNA OPCION: ","OPCION FUERA DE RANGO");
        //Pide al usuario que seleccione una opción para agregar más registros o salir.

    }while(op == 1);

    return tam;
}
//*****************************************************************************************************************


//*****************************************************************************************************************
void imprimirR(Tkey registros[], int tam)
{
    Tkey reg;
    int i;
    char *sexo;
```

```
    //Se declara una variable para almacenar un registro individual y una variable para iterar a través del arreglo de registros.
    //También se declara una variable para almacenar el sexo del estudiante.

    system("CLS");

    if (tam > 0) //Verifica si hay registros almacenados (tam > 0).
    {
        printf("%-10s %-14s %-15s %-20s %-20s %-10s %-10s\n", "ESTATUS", "MATRICULA", "NOMBRE", "APELLIDO PATERNO", "APELLIDO MATERNO", "EDAD"
, "GENERO\n");

        for (i = 0; i < tam; i++)  //Utiliza un bucle for para iterar a través del arreglo de registros.
        {

            if(registros[i].genero == 1)
            {

                sexo ="HOMBRE";

            }

            else
            {

                sexo ="MUJER";

            }

            if (registros[i].matricula != -1 && registros[i].matricula >= 300000 && registros[i].matricula <= 399999)
            {

                printf("%-10d %-14ld %-15s %-20s %-20s %-10d %-10s\n", registros[i].estatus, registros[i].matricula, registros[i].nombre,
registros[i].apellido_paterno, registros[i].apellido_materno, registros[i].edad, sexo);

            }

        }
            //Determina el sexo del estudiante en función del valor del campo genero en la estructura Talum.
            //Verifica si la matrícula del estudiante es válida (entre 300000 y 399999) antes de imprimir los datos del estudiante.

    }

    else
    {

        printf("INGRESE DATOS PARA PODER IMPRIMIR\n");

    }

    printf("\n");
    system("PAUSE");
}
//*************************************************************************************************************************


//*************************************************************************************************************************
void buscarR(Tkey registros[], int tam)
{
    system("CLS");
    long mat;
    int pos, id, i;
    char *sexo;
    int band = 0;
    //Se declara una variable para almacenar la matrícula a buscar (mat), la posición encontrada (pos), la matrícula ingresada (id), y una
variable para iterar (i).
    //También se declara una variable para almacenar el sexo del estudiante y una bandera para indicar si se encontró la matrícula (band).

    id = validaL(300000, 399999,"INGRESA LA MATRICULA QUE QUIERES BUSCAR: ","MATRICULA INVALIDA");
            //Utiliza la función validaL para solicitar y validar la matrícula a buscar, asegurando que esté en el rango válido.

    pos = busqSec(registros, tam, id); //Utiliza la función busqSec para buscar la matrícula en el arreglo de registros.

    if (pos != -1)  //Si se encuentra la matrícula (pos != -1), continúa con la impresión de los resultados. De lo contrario,
                    //imprime un mensaje indicando que no se encontró la matrícula.
    {
        system("CLS");
        printf("\nRESULTADOS ENCONTRADO.\n%-10s %-14s %-15s %-20s %-20s %-10s %-10s\n", "Eestatus", "matriculaCULA", "nombre", "APELLIDO
PATERNO", "APELLIDO MATERNO", "EDAD", "GENERO\n");

        if(registros[pos].genero == 1) //Determina el sexo del estudiante en función del valor del campo genero en la estructura Talum.
        {
            sexo ="HOMBRE";
        }
        else
        {
            sexo ="MUJER";
        }
```

```
        if (registros[pos].matricula != -1 && registros[pos].matricula >= 300000 && registros[pos].matricula <= 399999)
                        //Verifica si la matrícula del estudiante es válida (entre 300000 y 399999) antes de imprimir los datos del
estudiante.
        {
            printf("%-10d %-14ld %-15s %-20s %-20s %-10d %-10s\n", registros[pos].estatus, registros[pos].matricula, registros[pos].nombre,
registros[pos].apellido_paterno, registros[pos].apellido_materno, registros[pos].edad, sexo);
        }
    }
    else
    {
        printf("NO SE ENCONTRO LA MATRICULA\n");
    }

}
//*****************************************************************************************************************************


//*****************************************************************************************************************************
void eliminarR(Tkey registros[], int tam)
{
    system("CLS");

    long id;
    int pos, op;

    id = validaL(300000, 399999,"INGRESA LA MATRICULA QUE QUIERES ELIMINAR: ","MATRICULA INVALIDA");
            //Utiliza la función validaL para solicitar y validar la matrícula a eliminar, asegurando que esté en el rango válido.

    pos = busqSec(registros, tam, id);  //Utiliza la función busqSec para buscar la matrícula en el arreglo de registros.

    if (pos != -1) //Si se encuentra la matrícula (pos != -1), continúa con la eliminación. De lo contrario, imprime un mensaje indicando que
la matrícula no existe.
    {
        imprimirR(registros, tam);
        op = validaInt(1, 2,"1) ELIMINAR\n 2) NO ELIMINAR\n","OPCION FUERA DE RANGO");
        //Utiliza la función imprimirR para mostrar los datos del estudiante a eliminar.
        //Y solicita al usuario que confirme si desea eliminar o no al estudiante.

        if (op == 1) //Si el usuario confirma la opción de eliminar (op == 1), establece el estatus del estudiante a 0, marcando así al
estudiante como eliminado.
        {
            registros[pos].estatus = 0;
        }
    }
    else
    {
        printf("LA MATRICULA NO EXISTE\n");
    }
}
//*****************************************************************************************************************************
```