Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño F.I.A.D.

Campus Ensenada

Ramón Mejía Martínez 372099

Grupo: 932

Actividad 11

Programación Estructurada

Librerías en c, Cadenas, Funciones, Structs

Ensenada, Baja California a 1 de noviembre de 2023.

Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Ramón Mejía Martínez

Matrícula: 372099

Maestro: Pedro Núñez Yépiz

Actividad No. : 11

Tema - Unidad : Librerías en c, Cadenas, Funciones,Structs.

Ensenada Baja California a 1 de noviembre del 2023.

1. Introducción

En esta actividad se realizara un sistema de gestión de alumnos implementando el sistema de la curp con sus datos.

2. Competencia

Esta actividad tiene como objetivo el manejo cadenas, librerías y funciones, structs anidadas, métodos de ordenación y búsqueda para ir agregando herramientas que podrán ser usadas nuevamente en otros proyectos.

3. Fundamentos

Structs.

Los structs son un tipo de dato que permite combinar diferentes tipos de datos en una única estructura personalizada. Son muy útiles para el manejo de distintos datos relacionados entre sí.

Definir structs:

```
struct{
 tipo_dato variable1;
 tipo_dato variable2;.
.
.
.
};
```

Estas estructuras se pueden renombrar usando typedef:

```
typedef struct{
 tipo_dato campo1;
 tipo_dato campo2;.
.
.
.
}nombre;
```

Para acceder a cada campo se usa un punto entre el nombre de la struct y el campo al que se quiere acceder, por ejemplo:

```
 nombreStruct.campo1;
```

Se pueden crear arreglos de structs.

Librerías en C.

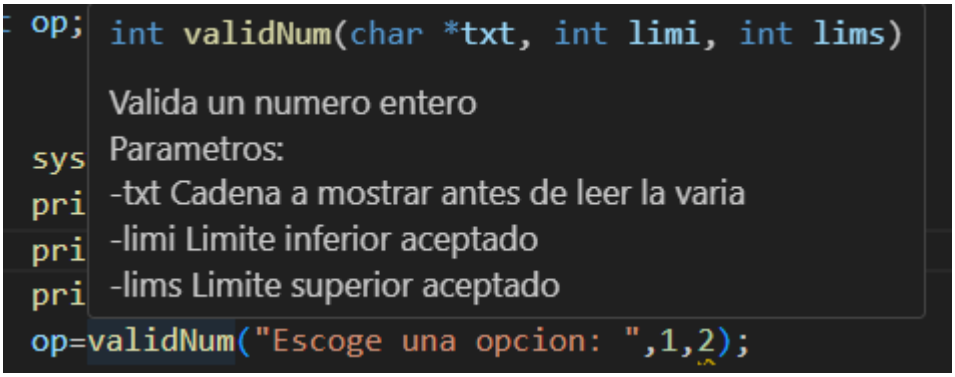
Las librerías en c son archivos o bloques de código con funciones especiales que podemos incluir en nuestro código importando sus funcionalidades.

Para la creación de nuestra propia librería podemos crear un archivo.h que contendrá todas las funciones que deseemos, este archivo puede contener los prototipos de las funciones e incluir los cuerpos en otro archivo también puede contener tanto prototipos como cuerpos. Para
importar estas librerías en nuestros archivos.c se hace de la siguiente manera:

#include “(ruta de la librería)”

Usando la palabra reservada #include con la que se agregan otras librerías, entre comillas escribimos la ruta de nuestra librería, ahora tendremos todas las funciones existentes en el archivo.h para nuestro archivo.c.

Para Visual Studio Code podemos escribir una descripción de la función con comentarios en el archivo.h y se mostrara en nuestro archivo.c para mejor entendimiento del código.



Cadenas.

En C no existen la variable tipo cadenas. Existen

- arreglos de caracteres
- punteros a carácter

Cuando se almacena una cadena de caracteres debe estar terminada por el carácter cuyo código es 0 (’\0’) ya que es lo que esperan encontrar las funciones de cadenas.

Las funciones de cadenas esperan una dirección de memoria (char *); como el nombre de un arreglo es la dirección donde se almacena el arreglo, las funciones de cadenas pueden recibir tanto un array como un puntero.

Para declarar una cadena se hace de la siguiente manera:

```
char identificador[10];
```

Usando la palabra reservada char, identificador y su dimensión, si la cadena lee un contenido menor a 10 por ejemplo “Casa” esto se ubicara de la siguiente forma.

0	1	2	3	4	5	6	7	8	9
‘C’	‘a’	‘s’	‘a’	‘\0’					

Las cadenas se pueden manejar de distintas formas ya sea como arreglos o con ciertas funciones, pero para esta practica se manejarán como arreglos exceptuando para leer la cadena.
Para leer una cadena se usa la función gets(identificador);

Para imprimir se puede usar scanf(“%s”,identificador); o imprimiendo carácter por carácter, para esto es necesario un estructura de control repetitiva:

```
for(i=0;identificador[i]!='\0';i++)
{
        printf(“%c”,identificador[i]);
}
```

Funciones.

Las funciones suelen encapsular una operación más o menos compleja de la que se deriva un resultado. Para ejecutar esta operación, las funciones pueden precisar la invocación de otras funciones (o incluso de ellas mismas como es el caso de las funciones recursivas).

Las funciones en un programa son entidades que dado un conjunto de datos (los parámetros), se les encarga realizar una tarea muy concreta y se espera hasta obtener el resultado. Lo idóneo es dividir tareas complejas en porciones más simples que se implementan como funciones. La división y agrupación de

tareas en funciones es uno de los aspectos más importantes en el diseño de un programa.

Las partes de las funciones son: Prototipo, se encuentra antes de la función main y es la definición de las funciones las cuales llevan, tipo de dato que devolverán, nombre y parámetros de entrada.

Ejemplo Prototipo:

```
int suma(int a,int b);
```

El int inicial es el tipo de dato que regresa, suma el nombre, a y b son los parámetros de los cuales se debe especificar qué tipo de datos son

Otra parte es la llamada a la función la cual se hace en la función main o dentro de otra función.

Ejemplo llamada:

```
resultado=suma(5,2);
```

```
resultado=suma(cal1,cal2);
```

Y por último el cuerpo de la función, esto va después de la función main, se debe poner el prototipo exactamente igual a excepción de que se quitan los puntos y comas y debajo se crea el bloque que contendrá el código del proceso que hará la función. Ejemplo cuerpo de la función:

```
int suma(int a,int b)
{
    int r;
    r=a+b;

    return r;

}
```

4. Procedimiento

1.

ACTIVIDAD 11
FUNCIONES y METODOS DE ORDENACION Y BUSQUEDA
ESTRUCTURAS Y LIBRERIAS (p2)

INSTRUCCIONES:
1.- Realiza un programa en C que utilice una librerías propias
(Funciones de validar números y cadenas)
2.- Realiza reporte de práctica
3.- Sube a Blackboard, programa, librería, y reporte de practica y PDF anexo con capturas y código

2.

PRÁCTICA 11

Realiza el programa que contenga el siguiente menú

M E N Ú
1.- Agregar
a) manual (1)
b) Automatico (100)
c) Regresar
2.- Eliminar Registro
3.- Buscar
4.- Ordenar
5.- Imprimir
6.- Archivo Texto

0.- Salir

El programa deberá poder almacenar en un arreglo (máximo 2,000 registros) los datos para generar el CURP la estructura debe contener 2 estructuras anidadas, nombre y fecha nacimiento y un campo donde se escribirá automáticamente el curp basado en los datos proporcionados

MENÚ DESCRIPCIÓN:

1.- Cargar: Se deberá agregar 100 registros en forma automáticamente y aleatorios (cuidar no se desborde Arreglo)

2.- Eliminar: La búsqueda se realizará por matrícula, Imprimir el registro encontrado en forma de registro y preguntar si quiere eliminar si o no. (Eliminado Lógico x campo status)

3.- Buscar: La búsqueda se realizará por matrícula, el programa deberá ser capaz de realizar la búsqueda secuencial o Binaria según sea el caso. Imprimir el registro encontrado en forma de registro.

4.- Ordenar: La ordenación será por MATRICULA usar función de ordenación más adecuada según sea el caso usar 2 métodos de ordenación y el programa decidirá cuál es el que *usará dependiendo del estado y tamaño de registros dentro del arreglo*.
Nota: (validar si el arreglo ya está ordenado no volver ordenar por el mismo campo)

5.- Imprimir: El programa deberá imprimir los datos del arreglo (solo registros activos) en forma de tabla en pantallas de 40 registros y presionando la tecla de continuar en cada uno de los casos.

6.- Archivo de Texto: El programa deberá generar un archivo de texto con los datos del arreglo (solo registros activos) formatear salida.

NOTA: forma de registro es de la siguiente manera:

MATRICULA : 300523
NOMBRE : YAREMI
NOMBRE2 : GHIZETH
AP PATERNO : GARCIA
AP MATERNO : GUERRERO
FECHA NAC : 03-04-2010
EDAD : 19
SEXO : MUJER
LUGAR NAC : BAJA CALIFORNIA SUR
CURP : GAGY030410MBCRRRA5

NOTA : Librería Propia, Usar funciones, no se permiten variables global

5. Resultados y conclusiones:

Structs anidadas:

```
typedef struct _name
{
    char appat[L];
    char apmat[L];
    char nombre[LN];
}StructName;

typedef struct _edad
{
    char anio[5];
    char mes[3];
    char dia[3];
}StructEdad;

typedef struct _alum
{
    int status;
    int matricula;
    StructName FullNombre;
    StructEdad FechaNacim;
    char estado[3];
    char sexo;
    int edad;
    char curp[18];
}StructAlum;
```

Main:

La función principal se usó para llamar al menú del programa y poner la semilla para los valores aleatorios.

```
int main()
{
    srand(time(NULL));
    menu();
    return 0;
}
```

Función menu:

- Se pregunta al usuario la opción que desea y se ejecutan las llamadas y preguntas extras necesarias al usuario.

```
void menu(void)
{
    StructAlum alumnos[STC];
    int op;
    int pos;
    int i=0,j;
    int txtband=1;
    bool orden=false;
    do
    {
        system("cls");
        op=msgs();
        system("cls");
        switch(op)
        {
            case 1:
                i=agregarMenu(alumnos,i);
                orden=false;
                break;
```

Caso 1:

- Se llama a un submenú para saber si se desea agregar un alumno de manera manual o 100 de manera aleatoria.

Función agregarMenu:

- Lee 100 alumnos aleatorios o 1 manual.
- Recibe todos los registros y el numero de registros existentes.
- Devuelve la cantidad de alumnos que se agregaron.

```c
//case 1
int agregarMenu(StructAlum alumnos[],int i)
{
    int op,s;
    int j;
    int band=1;
    int validmatri,matri;
    printf("Agregar alumno\n\n");
    printf("a) Manual\n");
    printf("b) Automatico\n");
    printf("c) Regresar\n");
    printf("Escoge una opcion: ");
    do
    {
        op=getch();
        if(op>96)
        {
            if(op<100)
            {
                printf("%c",op);
                getch();
                band=0;
            }
        }
    }while(band);
```

- Manual. Se llama a la función datos de la librería de la curp para leer los datos necesarios del usuario y se llama a la función curpGen para generar la curp.

```
case 97:
    system("cls");
    printf("Agregar alumno manual\n\n");
    alumnos[i].status=1;
    do···
    }while(validmatri!=-1);
    alumnos[i].matricula=matri;
    s=datos(alumnos[i].FullNombre.nombre,alumnos[i].FullNombre.appat,alumnos[i].F
    if(s==1)···
    else···
    curpGen(alumnos[i].FullNombre.nombre,alumnos[i].FullNombre.appat,alumnos[i].F
    alumnos[i].edad=calcularEdad(atoi(alumnos[i].FechaNacim.anio),atoi(alumnos[i]
    i++;
    break;
```

- Aleatorio.

```
case 98:
    if(i+100<=2000)
    {
        for(j=0;j<100;j++)
        {
            randAlum(alumnos,i++);
        }
        printf("\n\nSe agregaron 100 alumnos\n");
    }
    else
    {
        if(i!=2000)···
        else···
    }
    break;
```

Función randAlum:

- Genera todos los datos de un alumno de manera aleatoria.
- Recibe todos los registros y el numero de registros existentes.

```
void randAlum(StructAlum alumnos[],int i)
{
    int born[3];
    int s;
    int ri=1900,rf=2023,rango=(rf-ri+1);
    char estadoN[3];
    alumnos[i].status=1;
    alumnos[i].matricula=genMatri(alumnos,i);

    s=rand()%2;
    if(s)
    {
        alumnos[i].sexo='H';
    }
    else···

    if(s)
    {
        genH_Name(alumnos,i);
    }
    else···
```

```
s=rand()%2;
if(s)
{
    alumnos[i].sexo='H';
}
else…

if(s)
{
    genH_Name(alumnos,i);
}
else…

genAp(alumnos,i);

genEst(alumnos[i].estado);
strcopy(estadoN,alumnos[i].estado);

//fecha nacimiento
{
    bool leap=false;
    born[2]=(rand()%rango)+ri;
    itoa(born[2],alumnos[i].FechaNacim.anio,10);

    born[1]=(rand()%12)+1;
    itoa(born[1],alumnos[i].FechaNacim.mes,10);
    if(born[1]>=10)
    {
        itoa(born[1],alumnos[i].FechaNacim.mes,10);
    }
    else
```

Función genMatri:

- Genera una matrícula aleatoria que aun no exista en el registro.
- Recibe los registros y el número de registros.
- Devuelve la matricula generada.

```
//Generar Alumno
int genMatri(StructAlum alumnos[],int i)
{
    int num;
    do
    {
        num=(rand()%100000)+300000;
    }while(buscarAlumSec(alumnos,i,num)!=-1);
    return num;
}
```

Función genH_Name:

Función genM_Name:

Función genAp:

Estas tres funciones trabajan de la misma manera.

- Generan un nombre o apellido de manera aleatoria, según corresponda.
- Recibe los registros y el número de registros.

```
void genH_Name(StructAlum alumnos[],int i)
{
    char h_name[100][LN]={···
    int prob;
    prob=rand()%10;
    if(prob<=7)
    {
        strcopy(alumnos[i].FullNombre.nombre,h_name[rand()%100]);
    }
    else
    {
        strcopy(alumnos[i].FullNombre.nombre,h_name[rand()%100]);
        strConcat(alumnos[i].FullNombre.nombre,h_name[rand()%100]);
    }
}
```

Función genEst:

- Asigna de una lista de estados un estado elatorio..
- Recibe la cadena estado del registro.

Asigna de una lista de estados un estado elatorio.

```
void genEst(char estado[])
{
    int E;
    char estList[33][3]={
    "AS","BC","BS","CC","CS","CH","CM","CL","MX","DG",
    "GT","GR","HG","JC","EM","MN","MS","NT","NL","OC",
    "PL","QT","QR","SP","SL","SR","TC","TS","TL","VZ",
    "YN","ZS","NE"
    };


    E=rand()%33;

    estado[0]=estList[E][0];
    estado[1]=estList[E][1];
    estado[2]='\0';
}
```

Función calcularEdad:

- Calcula la edad actual del alumno según la fecha del sistema y su fecha de nacimiento.
- Recibe el año, mes y día de nacimiento del alumno.
- Devuelve la edad del alumno.

```
int calcularEdad(int anio,int mes,int dia)
{
    int edad;
    time_t t;
    t=time(NULL);
    struct tm *fecha;
    fecha=localtime(&t);

    edad=(fecha->tm_year+1900)-anio;
    if(mes>fecha->tm_mon+1)
    {
        edad--;
    }
    else
    {
        if(mes==fecha->tm_mon+1)
        {
            if(dia>fecha->tm_mday)
            {
                edad--;
            }
        }
    }
    return edad;
}
```

Caso 2:

Se busca el alumno a eliminar, se muestran sus datos y si el usuario lo desea se cambia su status a 0.

```
case 2:
    printf("Eliminar alumno\n\n");
    if(i>0)
    {

        pos=buscarAlumSec(alumnos,i,validNum("Ingresa la matricula del alumno que
        if(pos!=-1)
        {
            printf("Numero: %d\n",pos);
            printf("Lugar de nacimiento: %s\n",alumnos[pos].estado);
            printf("Apellido Paterno: %s\n",alumnos[pos].FullNombre.appat);
            printf("Apellido Materno: %s",alumnos[pos].FullNombre.apmat);
            printf("Nombre: %s",alumnos[pos].FullNombre.nombre);
            printf("Fecha de nacimiento: %d-%d-%d",atoi(alumnos[pos].FechaNacim.di
            printf("Edad: %d",alumnos[pos].edad);
            printf("Sexo: ");
            if(alumnos[pos].sexo=='H') ···
            else ···
            printf("Curp: %s\n",alumnos[pos].curp);
            printf("\nDesea eliminar a este alumno?\n");
            printf("1) Si\n");
            printf("2) No\n");
            if(validNum("Escoge una opcion: ",1,2)==1)
            {
                alumnos[pos].status=0;
                printf("\nSe elimino este alumno\n");
            }
        }
        else ···
    }
    else ···
    break;
```

Caso 3:

Se llama a un submenú para determinar qué tipo de búsqueda hacer.

```
case 3:
    if(i>0)
    {
        menuBusc(alumnos,i,orden);
    }
    else
    {
        printf("\nAun no hay alumnos registrados\n");
    }
    break;
```

Función menuBusc:

Si el vector esta ordenado se realiza la búsqueda binaria, si no lo esta se usa la búsqueda secuencial. Si encuentra al alumno muestra sus datos.

- Recibe los registros, el número de registros y la bandera que representa si los registros están ordenados.

```
void menuBusc(StructAlum alumnos[],int i,bool orden)
{
    int pos;
    printf("Busqueda\n\n");

    if(orden)
    {
        pos=buscarAlumBin(alumnos,i,validNum("Ingresa la matricula del alumno que desea buscar: ",300000,3999
        if(pos!=-1)
        {
            printf("Numero: %d\n",pos);
            printf("Lugar de nacimiento: %s\n",alumnos[pos].estado);
            printf("Apellido Paterno: %s\n",alumnos[pos].FullNombre.appat);
            printf("Apellido Materno: %s",alumnos[pos].FullNombre.apmat);
            printf("Nombre: %s",alumnos[pos].FullNombre.nombre);
            printf("Fecha de nacimiento: %d-%d-%d",atoi(alumnos[pos].FechaNacim.dia),atoi(alumnos[pos].FechaN
            printf("Edad: %d",alumnos[pos].edad);
            printf("Sexo: ");
            if(alumnos[pos].sexo=='H')
            {
                printf(" HOMBRE\n");
            }
            else···
            printf("Curp: %s\n",alumnos[pos].curp);
        }
        else···
    }
    else
    {
        pos=buscarAlumSec(alumnos,i,validNum("Ingresa la matricula del alumno que desea buscar: ",300000,3999
        if(pos!=-1)···
        else···
    }
```

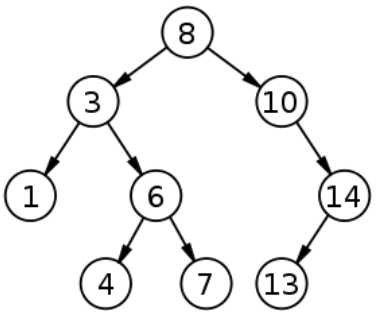
Función buscarAlumSec:

- Busca una matricula recorriendo todo el arreglo de alumnos.
- Recibe los registros, el numero de registros existentes y la matricula que se desea buscar.
- Regresa la posición de la matricula buscada si la encontró, si no la encontró regresa -1.

```
//case 3-1
int buscarAlumSec(StructAlum alumnos[],int i,int num)
{
    int j;
    for(j=0;j<i;j++)
    {
        if(num==alumnos[j].matricula)
        {
            return j;
        }
    }
    return -1;
}
```

Función buscarAlumBin:

Este método trabaja dividiendo el arreglo según si el numero a buscar es mayor o menor que el elemento que está a la mitad. Por ejemplo, si se desea buscar el 7 en el siguiente árbol binario, se compara con el 8, va a la izquierda, se compara con el 3, va a la derecha, se compara con el 6, va a la derecha, **se encontró el valor.



- Busca una matricula cuando el registro esta ordenado.
- Recibe los registros, el número de registros existentes y la matricula que se desea buscar.
- Regresa la posición de la matricula buscada si la encontró, si no la encontró regresa -1.

```
int buscarAlumBin(StructAlum alumnos[],int i,int num)
{
    int izq,drcha;
    int med;
    izq=0;
    drcha=i-1;
    while(izq<=drcha)
    {
        med=izq+(drcha-izq)/2;
        if(alumnos[med].matricula==num)
        {
            return med;
        }

        if(alumnos[med].matricula<num)
        {
            izq=med+1;
        }
        else
        {
            drcha=med-1;
        }
    }
    return -1;
```

Caso 4:

Si el número de elementos es mayor o igual a 1000 se usa el método mergeSort, si no se usa el heapSort para ordenar los registros.

```
case 4:
    if(!orden)
    {
        if(i>=1000)
        {
            mergeSort(alumnos,0,i-1);
        }
        else
        {
            heapSort(alumnos,i);
        }
        printf("Se ordenaron los registros\n");
        orden=true;
    }
    break;
```

```
//MergeSort
void mergeSort(StructAlum alumnos[],int izq,int derch)
{
    int med;

    if(izq<derch)
    {
        med=izq+(derch-izq)/2;
        mergeSort(alumnos,izq,med);
        mergeSort(alumnos,med+1,derch);
        fusionMergeSort(alumnos,izq,med,derch);
    }
}
```

```
void fusionMergeSort(StructAlum alumnos[],int izq,int medio,int derch)
{
    int i,j,k;
    int n1=medio-izq+1;
    int n2=derch-medio;
    StructAlum Lf[n1], R[n2];

    for (i=0;i<n1;i++)···
    for (j=0;j<n2;j++) ···

    i=0;
    j=0;
    k=izq;
    while (i<n1)
    {
        if(j<n2)
        {
            if(Lf[i].matricula<=R[j].matricula)
            {
                alumnos[k]=Lf[i];
                i++;
            }
            else
            {
                alumnos[k]=R[j];
                j++;
            }
            k++;
```

```
void heapSort(StructAlum alumnos[], int n)
{
    int i;
    for (i=n/2-1;i>=0;i--)
    {
        adjustHeap(alumnos,n,i);
    }

    for (i=n-1;i>0;i--)
    {
        swap(&alumnos[0],&alumnos[i]);
        adjustHeap(alumnos,i,0);
    }
}
```

```
void adjustHeap(StructAlum alumnos[],int n,int raiz)
{
    int may=raiz;
    int izq =2 *raiz+1;
    int derch =2*raiz+2;

    if(izq<n)
    {
        if(alumnos[izq].matricula>alumnos[may].matricula)
        {
            may=izq;
        }
    }

    if(derch<n)
    {
        if(alumnos[derch].matricula>alumnos[may].matricula)
        {
            may=derch;
        }

    }

    if(may!=raiz)
    {
        swap(&alumnos[raiz],&alumnos[may]);
        adjustHeap(alumnos,n,may);
    }
}
```

Caso 5:

Manda a imprimir todos los registros uno por uno.

```
case 5:
    printf("Imprimir alumnos\n\n");
    if(i>0)
    {
        printf("| No  | Est | Matri  | Apellido Paterno | Apellido Materno | No
        for(j=0;j<i;j++)
        {
            printf("| %3d ",j);
            printAlum(alumnos[j]);
            if((j+1)%40==0)
            {
                printf("\n");
                system("pause");
                printf("\n");
            }
        }
    }
    else…
    break;
```

Función printAlum:

- Imprime la información de un solo alumno en forma de tabla.
- Recibe los datos del alumno.

```
//case5
void printAlum(StructAlum alumnos)
{
    printf("| %s  | %d | %-16s | %-17s| %-19s | %3d |  %c  |",alumnos.estado,alumnos.m
    printf(" %s |\n",alumnos.curp);
}
```

Caso 6:

Se llama a una función para generar el archivo con de los registros

```
case 6:
    if(i>0)
    {
        txtband=archivoTxt(alumnos,i,txtband);
        printf("Se genero un archivo de texto de los registros\n");
    }
    else
    {
        printf("\nAun no hay alumnos registrados\n");
    }
    break;
```

Función archivoTxt:

- Crea o sobrescribe los datos de todos los registros en un archivo de texto.
- Recibe los registros, el número de registros existentes y la bandera de que ya se ha llenado el archivo una vez (no supe cómo hacer para identificar que ya existía este archivo desde antes de la ejecución del programa así que se agrega la primera vez junto a los datos de la última ejecución si es que el archivo ya existe y las siguientes veces se sobrescribe de manera correcta).
- Regresa un 0, indicando que ya existe el archivo de texto.

```
int archivoTxt(StructAlum alumnos[],int i,int band)
{
    int j;
    FILE *pa;
    if(band)
    {
        pa=fopen("registros.txt","a");
        char str[119]={"| No  | Est | Matri  | Apellido Paterno | Apellido Materno
        fprintf(pa,"%s",str);

        for(j=0;j<i;j++)
        {
            if(alumnos[j].status)
            {
                fprintf(pa,"| %4d ",j);
                fprintf(pa,"| %s  | %d | %-16s | %-17s| %-19s | %3d |  %c  |",alum
                fprintf(pa," %s |\n",alumnos[j].curp);
            }
        }
    }
    else
    {
        pa=fopen("registros.txt","w");
        char str[119]={"| No  | Est | Matri  | Apellido Paterno | Apellido Materno
        fprintf(pa,"%s",str);

        for(j=0;j<i;j++)
        {
            if(alumnos[j].status)
            {
```

Conclusión:

Al usar código que no estaba pensado para ser reutilizado se puede complicar al querer reorganizar todo, algunas funciones tuvieron que ser cambiadas un poco, aunque en general todo funciono bien, el único problema fue que la cadena del struct de los estados perdía su contenido por alguna razón.

Direccion Est Dircopia Estcopia Direccion Est Dircopia Estcopia

```
00000000005F3B8E QT 00000000005F3950 QT || 00000000005F3B8E  00000000005F3950 QT
00000000005F3C2A MX 00000000005F3950 MX || 00000000005F3C2A MX 00000000005F3950 MX
00000000005F3CC6 MX 00000000005F3950 MX || 00000000005F3CC6 MX 00000000005F3950 MX
00000000005F3D62 GT 00000000005F3950 GT || 00000000005F3D62  00000000005F3950 GT
00000000005F3DFE MS 00000000005F3950 MS || 00000000005F3DFE  00000000005F3950 MS
00000000005F3E9A GR 00000000005F3950 GR || 00000000005F3E9A  00000000005F3950 GR
00000000005F3F36 BC 00000000005F3950 BC || 00000000005F3F36  00000000005F3950 BC
00000000005F3FD2 HG 00000000005F3950 HG || 00000000005F3FD2 HG 00000000005F3950 HG
00000000005F406E GR 00000000005F3950 GR || 00000000005F406E GR 00000000005F3950 GR
00000000005F410A GR 00000000005F3950 GR || 00000000005F410A GR 00000000005F3950 GR
```

Esto pasaba al entrar en la función que generaba la curp, para solucionar esto copiaba la cadena del estado y lo volvía a asignar al salir de la generación de la curp, ya que si no se cortaba la curp en el estado.

Pasado un rato esto se soluciono y dejo de perder la información. Aunque la razón de porque se soluciono es me es desconocida.

```
Escoge una opcion: b
 00000000005F1C41 SL 00000000005F1A00 SL || 00000000005F1C41 SL 00000000005F1A00 SL
 00000000005F1CE1 CS 00000000005F1A00 CS || 00000000005F1CE1 CS 00000000005F1A00 CS
 00000000005F1D81 EM 00000000005F1A00 EM || 00000000005F1D81 EM 00000000005F1A00 EM
 00000000005F1E21 CC 00000000005F1A00 CC || 00000000005F1E21 CC 00000000005F1A00 CC
 00000000005F1EC1 MX 00000000005F1A00 MX || 00000000005F1EC1 MX 00000000005F1A00 MX
```

6. Anexos

PDF:

https://drive.google.com/file/d/1jlfOQfGaVHj7_x7Ii7S-Sz01Clr4g0Nm/view?usp=sharing

Repositorio:

https://github.com/Muners24/Programacion-Estructurada

7. REFERENCIAS

Diseño de algoritmos y su codificación en lenguaje C

Corona, M.A. y Ancona, M.A. (2011)..

España: McGraw-Hill.

ISBN: 9786071505712

Programación estructurada a fondo:implementación de algoritmos en C

:Pearson Educación.Sznajdleder, P. A. (2017)..

Buenos Aires,Argentina: Alfaomega

Como programar en C/C++

H.M. Deitel/ P.J. Deitel

Segunda edición

Editorial: Prentice Hall.

ISBN:9688804711

Programación en C.Metodología, estructura de datos y objetos

Joyanes, L. y Zahonero, I. (2001)..

España:McGraw-Hill.

ISBN: 8448130138

De Computadores, P., & Olave, T. A. (s/f). *Algoritmos de Búsqueda y Ordenamiento*.

Utfsm.cl. Recuperado el 9 de octubre de 2023, de https://www.inf.utfsm.cl/~noell/IWI-131-p1/Tema8b.pdf