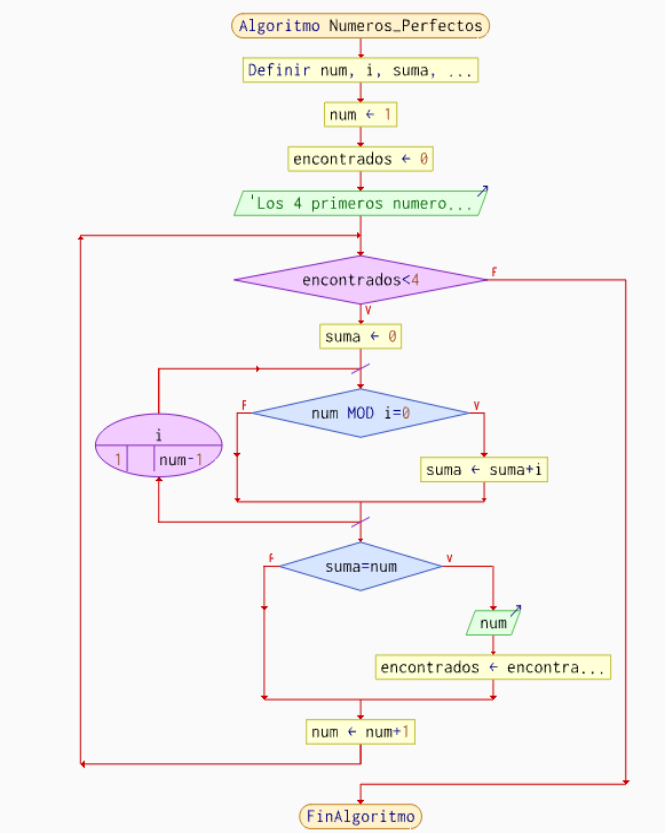
1. DATOS INFORMATIVOS

Carrera: Electronica y Automatization Asignatura: Fundamentos de Programacion

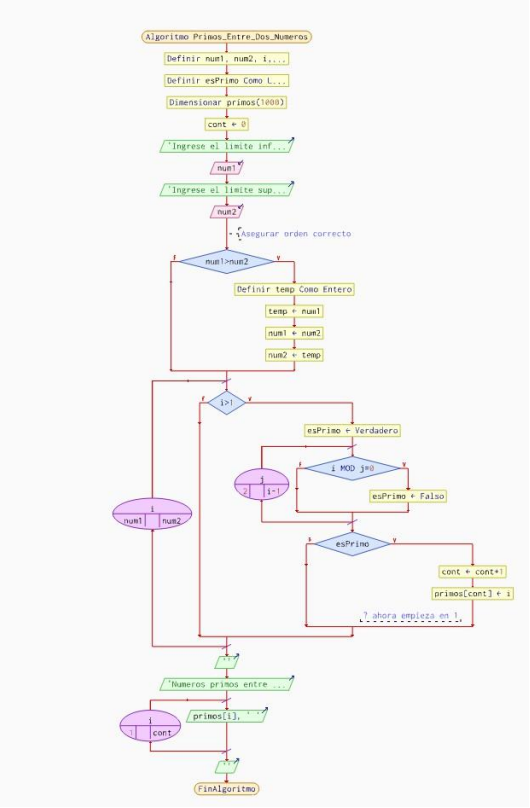
Integrantes: Ederson Gualoto

Un número perfecto es un número natural que es igual a la suma de sus divisores propios positivos, sin incluirse él mismo. Por ejemplo, el 6 es un número perfecto, porque sus divisores propios son 1, 2 y 3; y 6=1+2+36 = 1 + 2 + 36=1+2+3. El siguiente número perfecto es 28 = 1 + 2 + 4 + 7 + 14. Se pide: 1. Programe la función esperfecto, que reciba como argumento un número natural y devuelva un valor 1 si el número es perfecto, y 0 en caso contrario. 2. Desarrolle lafunción principal del programa que calcule los 4 primeros números perfectos y los muestre por pantalla.

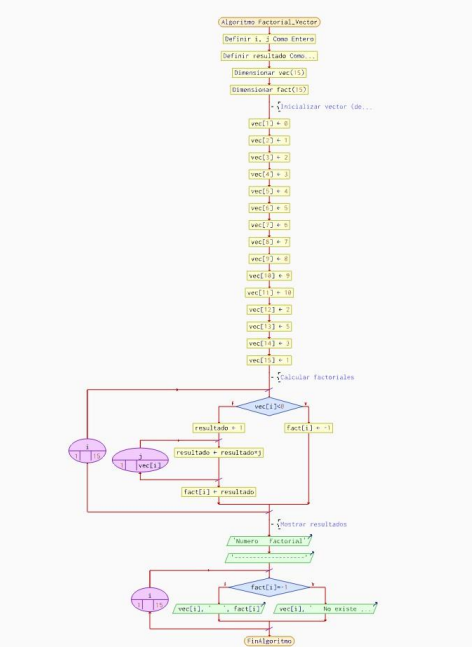



```c
#include <stdio.h>
int esperfecto(int n)
{
    int i, suma = 0;

    para (i = 1;  Yo < n;  Yo++)
    {
        if (n % i == 0)
            suma += i;
    }
    if (suma == n)
        Retorno 1;
    si no,
        retorno 0;
}
int main(void)
{
    int num = 1;
num ¿¿Perfecto?  encontrados Se imprime
1 No 0 No
6 S í 1 6
28 S í 2 28
496 S í 3 496
8128 Sí 4 8128
I ¿28 % i == 0?  SUMA
1 Sí 1
2 Sí 3
4 Sí 7
7 S í 14
14 S í 28
int encontrados = 0;
printf("Los 4 primeros números perfectos son:\n");
mientras (encontrados < 4)
{
}
}
}
si (esperfecto(num))
{
}
printf("%d\n", num);
encontrados++;
num++;
}
}
```

Problema 3.1.2 Números primos. Realice un programa que resuelva adecuadamente los siguientes apartados: 1. Programe la función esprimo, que recibe como argumento un número entero, y devuelve un valor 1 si el número es primo y 0 en caso contrario. 2. Almacene en unvector todos los números primos comprendidos entre dos números introducidos por teclado y luego imprima dicho vector.



Programe la función calc_fact, que recibe como argumento un número entero, y devuelva el valor de la factorial. A continuación, use esta función en un programa que, dado un vector de 15 números enteros vec, calcule un vector fact con sus factoriales y lo muestre por pantalla.



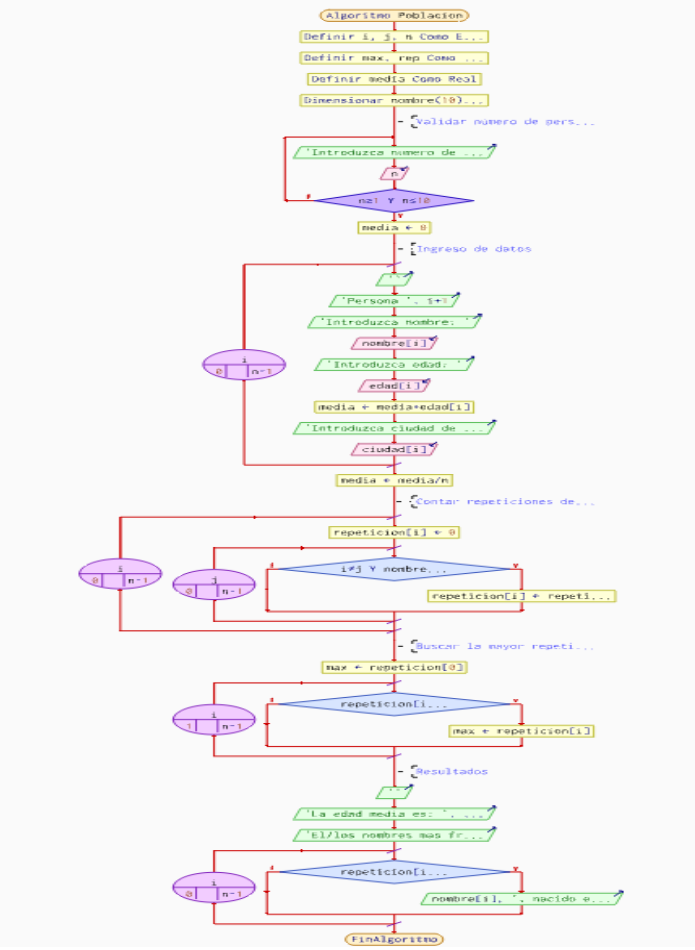
2. *Realice el programa principal que declare una tabla de estructuras de dimensión 10 para almacenar la información sobre personas (se ha supuesto que el número de personas no será mayor de 10). A continuación debe pedir por teclado el número de personas para introducir y después los datos de cada una de ellas. Tras ello, calcule el nombre que se repita más en los datos introducidos y la media de edad de todas las personas.*

   Se puede usar la función de librería:

   ```
   int strncmp(const char * s1,const char * s2);
   ```

   La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según la cadena apuntada por s1 es mayor, igual, o menor que la cadena s2.

Tras la lectura de todos los datos de personas se calcula el número de veces que se repite el nombre, actualizándolo en el campo *rep*, y almacenando en la variable *max* el mayor número de veces que se repite hasta el momento un nombre.



```c
#include <stdio.h>
#include <string.h>

#define MAX 10
#define STR 100

typedef struct {
    char nombre[STR];
    int edad;
    char ciudad[STR];
    int rep;
} Poblacion;

int main() {

    int i, j, n;
    int maxRep = 0;
    float media = 0;

    Poblacion v[MAX];

    /* Validar número de personas */
    do {
        printf("Introduzca numero de personas (1 a 10): ");
        scanf("%d", &n);
    } while (n < 1 || n > 10);

    /* Ingreso de datos */
    for (i = 0; i < n; i++) {

        printf("\nPersona %d\n", i + 1);

        printf("Introduzca nombre: ");
        scanf("%s", v[i].nombre);

        printf("Introduzca edad: ");
        scanf("%d", &v[i].edad);
        media += v[i].edad;

        printf("Introduzca ciudad de nacimiento: ");
        scanf("%s", v[i].ciudad);
    }
    }

    media = media / n;

    /* Contar repeticiones de nombres */
    for (i = 0; i < n; i++) {
        v[i].rep = 0;
        for (j = 0; j < n; j++) {
            if (i != j && strcmp(v[i].nombre, v[j].nombre) == 0) {
                v[i].rep++;
            }
        }
    }

    /* Buscar máxima repetición */
    maxRep = v[0].rep;
    for (i = 1; i < n; i++) {
        if (v[i].rep > maxRep) {
            maxRep = v[i].rep;
        }
    }

    /* Resultados */
    printf("\nLa edad media es: %.2f\n", media);
    printf("El/los nombres mas frecuentes son:\n");

    for (i = 0; i < n; i++) {
        if (v[i].rep == maxRep) {
            printf("%s, nacido en %s\n", v[i].nombre, v[i].ciudad);
        }
    }

    return 0;
}
```

Problema 4.2 Reconocimiento de caracteres.

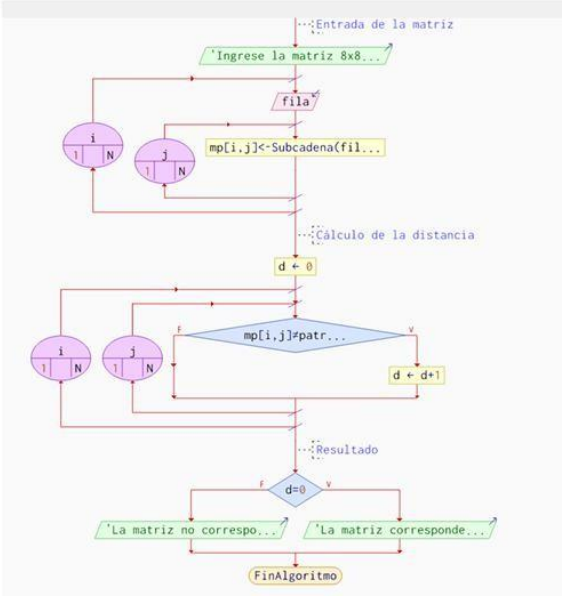
Se pretende escribir un programa para reconocer caracteres a partir de un mapa de puntos. El mapa de puntos describe la forma de un carácter como una matriz de unos y ceros de 8 × 8 celdas (véase figura 4.1). Se dispone además de una tabla de estructuras de tipo `struct letras`*, que puede suponer convenientemente creada e inicializada, y que contiene la descripción de las 27 letras del alfabeto, tal como se describe en el ejemplo.*

```
struct letras
{
   char cod_ASCII;   /* Letra a la que corresponde la matriz  de puntos*/
   int mptos[8][8];  /* Matriz de puntos del carácter */
};
struct letras tab_let[27];
```

0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	1

Figura 4.1: Representación del carácter *a* minúscula

Escriba la función `busca_caracter`*, la cual recibe una matriz de 8 × 8 enteros que contiene el mapa de puntos de la letra que se quiere identificar (parámetro mp) y la tabla de estructuras de tipo letras que contiene la descripción de las 27 letras del alfabeto (*`tab_let`*). La función debe devolver el código ASCII del carácter que más se parezca.*



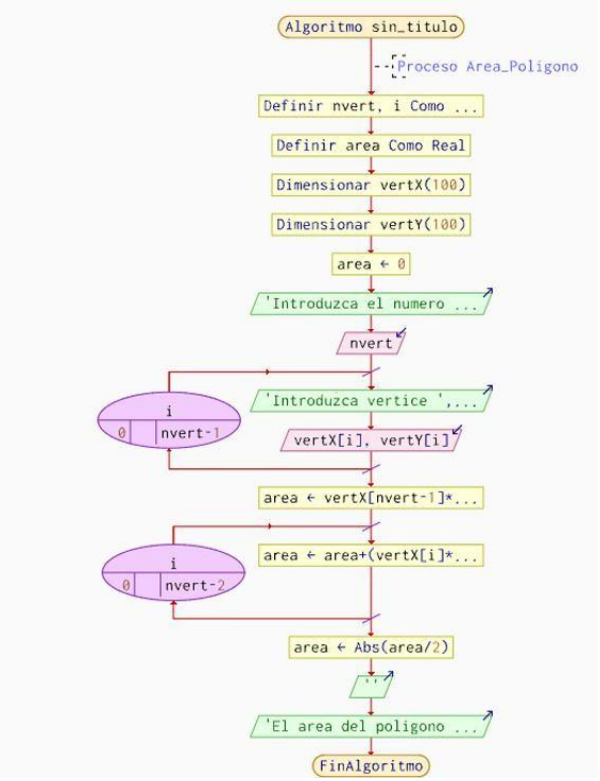
```c
#include <stdio.h>

#define N 8

typedef struct {
    char cod_ASCII;
    char mpost[N][N];
} letras;

/* Función para calcular la distancia entre dos matrices */
int distancia(const char a[N][N], const char b[N][N]) {
    int d = 0;
    int i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (a[i][j] != b[i][j]) {
                d++;
            }
        }
    }
    return d;
}

int main() {

    char mp[N][N];
    letras a_min;
    int i, j, d;

    /* Código ASCII */
    a_min.cod_ASCII = 'a';

    /* Patrón de la letra 'a' */
    char patron_a[N][N] = {
        {'0','0','#','#','#','#','0','0'},
        {'0','#','0','0','0','0','#','0'},
        {'0','0','0','0','0','0','#','0'},
        {'0','0','#','#','#','#','0','0'},
        {'0','#','0','0','0','0','#','0'},
        {'#','0','0','0','0','0','#','0'},
        {'0','#','0','0','0','0','#','0'},
        {'0','0','#','#','#','#','0','#'}
    };
    patronA[4,7] ← "0"
```

```c
};

/* Copiar patrón a la estructura */
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a_min.mpost[i][j] = patron_a[i][j];
    }
}

/* Ingreso de la matriz */
printf("Ingrese la matriz 8x8 usando SOLO 0 y #:\n");
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        scanf(" %c", &mp[i][j]);
    }
}

/* Comparación */
d = distancia(mp, a_min.mpost);

if (d == 0) {
    printf("\nLa matriz corresponde a la letra 'a' minuscula\n");
} else {
    printf("\nLa matriz NO corresponde a la letra 'a' minuscula\n");
}

return 0;
```

https://onlinegdb.com/lgQA3Qwb5

Un polígono es una figura geométrica cerrada delimitada por segmentos rectos (aristas). En este ejercicio se usa la estructura tipo polígono para almacenar la información de un polígono, donde en vez de almacenarse sus aristas se almacenan sus vértices, como se describe a continuación: struct punto { float x; float y; }; struct poligono { int nvert; /* Coordenada x */ /* Coordenada y */ /* Nro de vértices */ struct punto vert[100]; /* Vector de vértices */ }; De este modo, un polígono de la figura 4.2 quedaría representado mediante una estructura tipo struct poligono, cuyo miembro contiene el número de vértices del polígono (5), y vert es un vector de estructuras que contiene las coordenadas de los vértices con dimensión 100.

```c
#include <stdio.h>
#include <math.h>

#define MAX_VERTICES 100

/* ===== ESTRUCTURAS ===== */
struct punto {
    float x;
    float y;
};

struct poligono {
    int nvert;              // Número de vértices
    struct punto vert[MAX_VERTICES]; // Vector de vértices
};

/* ===== FUNCIONES ===== */

// Función para leer los vértices del polígono
void leerPoligono(struct poligono *p) {
    printf("Ingrese el numero de vertices del poligono (máx %d): ", MAX_VERTICES);
    scanf("%d", &p->nvert);

    if (p->nvert < 3 || p->nvert > MAX_VERTICES) {
        printf("Número de vértices inválido.\n");
        return;
    }

    for(int i = 0; i < p->nvert; i++) {
        printf("Vértice %d - coordenada x: ", i+1);
        scanf("%f", &p->vert[i].x);
        printf("Vértice %d - coordenada y: ", i+1);
        scanf("%f", &p->vert[i].y);
    }
}

// Función para imprimir los vértices del polígono
void mostrarPoligono(struct poligono p) {
    printf("\nLos vértices del polígono son:\n");
    for(int i = 0; i < p.nvert; i++) {
        printf("Vértice %d: (%.2f, %.2f)\n", i+1, p.vert[i].x, p.vert[i].y);
    }
}

// Función para calcular el perímetro del polígono
float calcularPerimetro(struct poligono p) {
    float perimetro = 0.0;
    for(int i = 0; i < p.nvert; i++) {
        int siguiente = (i + 1) % p.nvert; // Se conecta el último con el primero
        float dx = p.vert[siguiente].x - p.vert[i].x;
        float dy = p.vert[siguiente].y - p.vert[i].y;
        perimetro += sqrt(dx*dx + dy*dy); // distancia entre dos puntos
    }
    return perimetro;
}

/* ===== PROGRAMA PRINCIPAL ===== */
int main() {
    struct poligono miPoligono;

    // Leer vértices
    leerPoligono(&miPoligono);

    // Mostrar los vértices
    mostrarPoligono(miPoligono);

    // Calcular e imprimir perímetro
    float perimetro = calcularPerimetro(miPoligono);
    printf("\nEl perimetro del poligono es: %.2f\n", perimetro);

    return 0;
}
```



https://onlinegdb.com/plyEXsx6a

Se pretende hacer un pequeño juego de palabras por ordenador en lenguaje C que muestre en pantalla los caracteres que aparecen en la figura 4.3 de fin de juego. Considere la siguiente estructura: #define N 10 #define M 12 struct mensaje { char game_over[N]; char se_acabo[M]; }; typedef struct mensaje men; (En la figura 4.3 se muestra en pantalla el siguiente efecto visual:)

G

GA

GAM

GAME

GAME

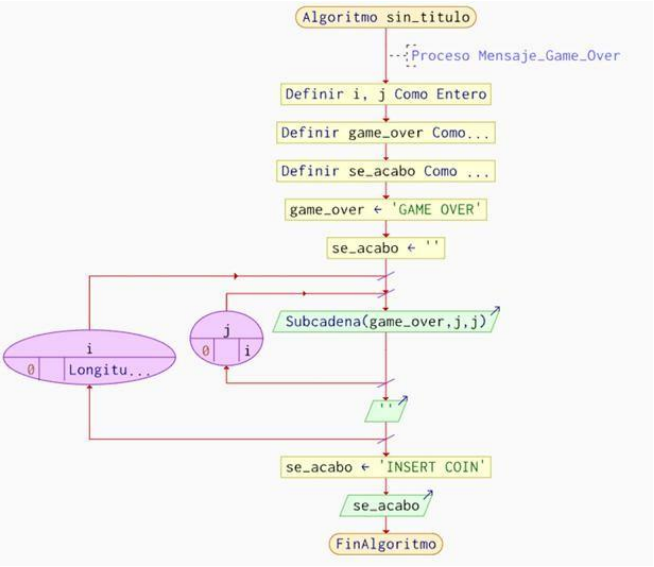
GAME O

GAME OV

GAME OVE

GAME OVER

INSERT COIN




```c
#include <stdio.h>
#include <string.h>

#define N 10
#define M 12

// Estructura mensaje con dos cadenas
struct mensaje {
    char game_over[N];
    char se_acabo[M];
};

typedef struct mensaje men;

// Función que imprime los primeros n caracteres de game_over
void prn_cad(men *m, int n) {
    int i = 0;
    while (i < n) {  // imprime hasta n-1
        printf("%c", m->game_over[i]);
        i++;
    }
    printf("\n"); // salto de línea al final
}

int main(void) {
    // Inicializar estructura
    men fin;
    strcpy(fin.game_over, "GAME OVER");
    strcpy(fin.se_acabo, "");  // inicializamos vacío

    // Imprimir progresivamente "GAME OVER"
    for (int i = 1; i <= strlen(fin.game_over); i++) {
        prn_cad(&fin, i);
    }

    // Asignar "INSERT COIN" a se_acabo
    strcpy(fin.se_acabo, "INSERT COIN");

    // Mostrar mensaje final
    printf("%s\n", fin.se_acabo);

    return 0;
}
```

https://onlinegdb.com/kDJ8Fe4RH

En C, las cadenas son arreglos de caracteres terminados en '\0'. Para leer texto de forma segura, se recomienda usar fgets() en lugar de scanf() y eliminar el salto de línea con strcspn(). Estructura Estudiante (basada en el documento):

Trabajaremos con estudiantes.txt. Cada línea se leerá con fgets() y se convertirá a un Estudiante con sscanf(), usando el mismo patrón del documento base.}

La operación Crear solicita datos y los guarda en modo "a" (añadir) usando fprintf(). Antes de agregar, es buena práctica validar que el ID no exista.

Leer consiste en abrir el archivo en modo "r" y mostrar los registros. Se lee línea por línea con fgets().

Buscar es una consulta específica: recorrer el archivo y comparar IDs con strcmp().

En archivos de texto secuenciales, una estrategia práctica para actualizar es: leer el archivo original y escribir uno temporal. Si la línea corresponde al ID buscado, se escribe la versión actualizada; de lo contrario, se copia la línea tal cual. Al final, se reemplaza el archivo original por el temporal. Este patrón es análogo al ejemplo del documento base para eliminar: usar tmp.txt, luego remove() y rename().

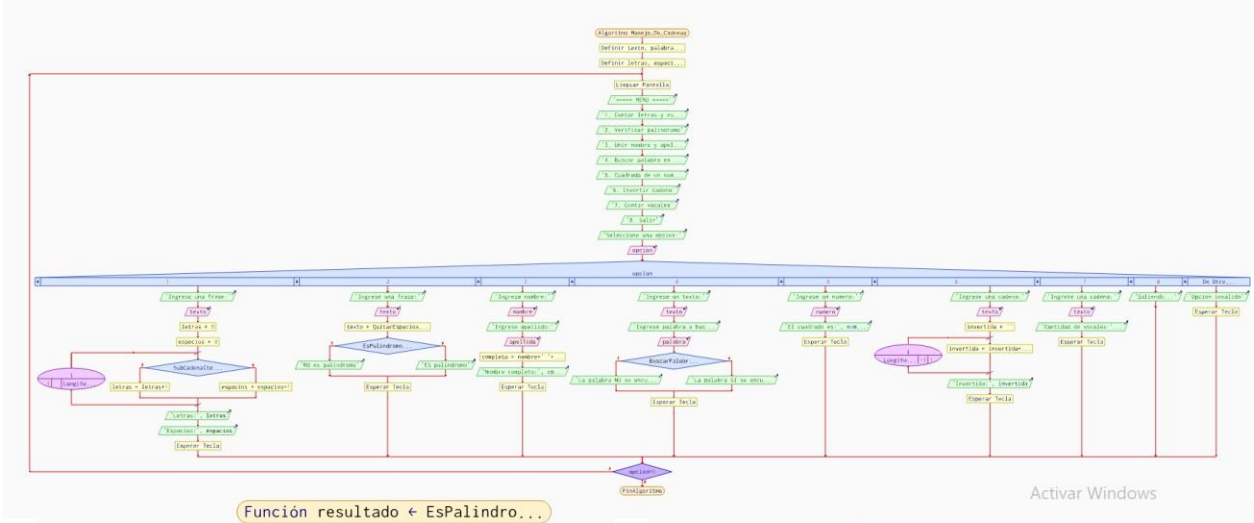
Eliminar (baja física) se implementa igual que en el ejemplo del documento base: se lee el archivo original y se reescribe un temporal omitiendo el registro a eliminar. Luego se reemplaza el original.

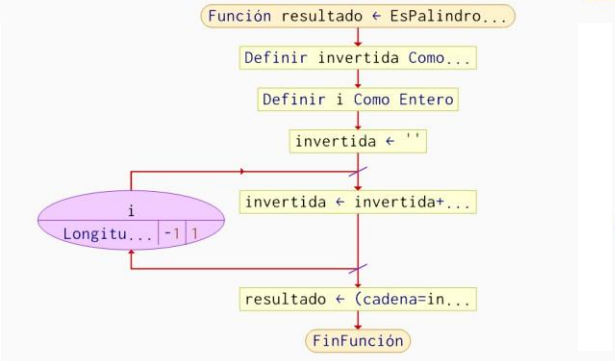
Integra las funciones en un ciclo do-while. Para primer nivel, un menú por consola es suficiente.

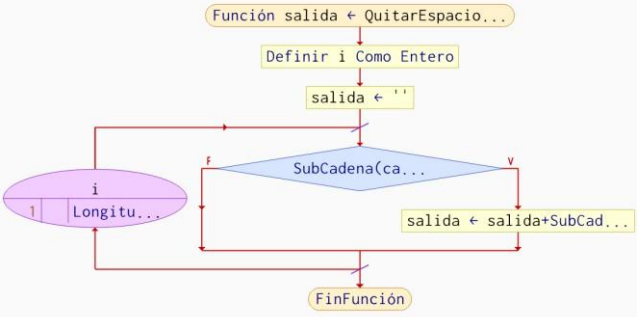
```
                eliminado = 1;
            }
        }
    }

    fclose(f);
    Fclose (temp);
    eliminar(ARCHIVO_EST);
    renombrar ("tmp.txt", ARCHIVO_EST);

    si (eliminado)
        printf("Registro eliminado.\n");
    si no,
        printf("ID no encontrado.\n");
}

/* ===== MENÚ ===== */
Menú INT Menú() {
    char OP[10];
    printf("\n=== CRUD ESTUDIANTES ===\n");
    printf("1. Agregar\n");
    Printf("2. Listar\n");
    printf("3. Consultor");
    Printf("4. Actualizar\n");
    Printf("5. Eliminar\n");
    printf("0. Salir\n");
    printf("Opción: ");
    fgets(OP, Sizeof(OP), Stdin);
    return atoi(op);
}

Int Main() {
    int op;
    do {
        OP = menú();
        Switch (op) {
            caso 1: agregarEstudiante();  Pausa;
            caso 2: listarEstudiantes();  Pausa;
            caso 3: consultarEstudiante();  Pausa;
            caso 4: actualizarEstudiante();  Pausa;
            caso 5: eliminarEstudiante();  Pausa;
            caso 0: printf("Saliendo...\n");  Pausa;
            default: printf("Opción inválida.\n");
        }
    } mientras (op != 0);

    retorno 0;
}
```

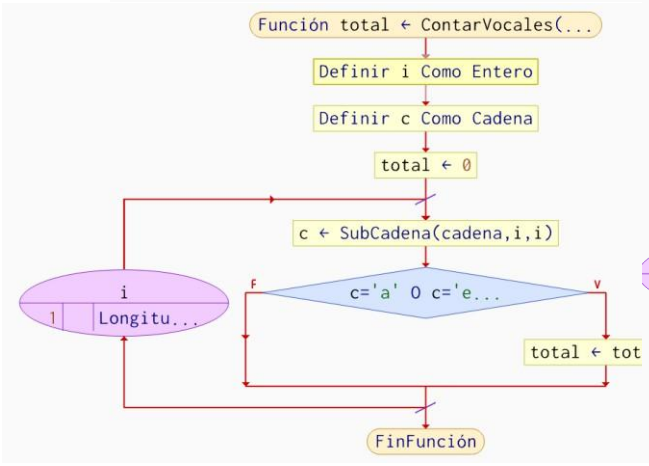
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define ARCHIVO_EST "estudiantes.txt"

/* ===== ESTRUCTURA ===== */
Typedef struct {
    Char ID[20];
    Char apellidos[50];
    nombres de caracteres[50];
    int edad;
} Estudiante;

/* ===== UTILIDADES ===== */
void limpiarNuevaLinea(char *s) {
    s[strcspn(s, "\n")] = '\0';
}

void leerCadena(const char *msg, char *dest, int tam) {
    printf("%s", MSG);
    fgets(dest, tam, stdin));
    limpiarNuevaLinea(dest);
}

int leerEntero(const char *msg) {
    Char Buf[50];
    char *endptr;
    val largo ;

    mientras (1) {
        printf("%s", MSG);
        fgets(buf, sizeof(buf), stdin);
        val = strtol(buf, & endptr, 10);

        si (endptr != buf && *endptr == '\n') {
            return (int)val;
        }
        printf("Entrada inválida. Intente nuevamente.\n");
    }
}

int parsearEstudiante(const char *linea, Estudiante *e) {
    return sscanf(linea, "%19[^;];%49[^;];%49[^;];%d",
                  e->id, e->apellidos, e->nombres, &e->edad) == 4;
}

/* ===== CRUD ===== */
int existeId(const char *idBuscado) {
    ARCHIVO *f = fopen(ARCHIVO_EST, "r");
    si (!f) retorno 0;

    Char Linea[200];
    Estudiante e;

    mientras que (fgets(linea, sizeof(linea), f)) {
        si (parsearEstudiante(linea, &e) && strcmp(e.id, idBuscado) == 0) {
            fclose(f);
            Retorno 1;
        }
    }
    fclose(f);
    retorno 0;
}
```

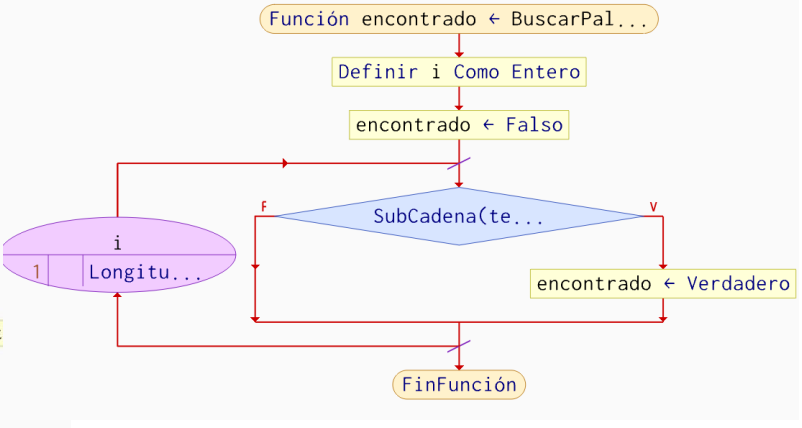
```
}

void agregarEstudiante() {
    Estudiante e;

    leerCadena("ID: ", e.id, tamaño(e.id));
    si (existeId(e.id)) {
        Printf("El ID ya existe.\n");
        Regreso;
    }

    leerCadena ("Apellidos: ", e.apellidos, tamaño de(e.apellidos));
    leerCadena("Nombres: ", e.nombres, tamaño(e.nombres));
    e.edad  = leerEntero("Edad: ");

    ARCHIVO *f = fopen(ARCHIVO_EST, "a");
    si (!f) {
        printf("Error al abrir archivo.\n");
        Regreso;
    }

    fprintf(f, "%s;%s;%s;%d\n", e.id, e.Apellidos, E.nombres, por ejemplo.edad);
    fclose(f);
    printf("Estudiante agregado correctamente.\n");
}

void listarEstudiantes() {
    ARCHIVO *f = fopen(ARCHIVO_EST, "r");
    si (!f) {
        printf("No existen registros.\n");
        Regreso;
    }

    Char Linea[200];
    Estudiante e;

     printf("\n%-10s %-20s %-20s %-5s\n", "ID", "APELLIDOS", "NOMBRES", "EDAD");
    printf("-----------------------------------------------------------\n");

    mientras que (fgets(linea, sizeof(linea), f)) {
        si (parsearEstudiante(linea, &e)) {
            printf("%-10s %-20s %-20s %-5d\n",
                   e.id, e.Apellidos, E.nombres, por ejemplo.edad);
        }
    }
    fclose(f);
}

void consultarEstudiante() {
    Char ID[20];
    Estudiante e;

    leerCadena("ID a buscar: ", id, sizeof(id));

    ARCHIVO *f = fopen(ARCHIVO_EST, "r");
    si (!f) {
        printf("Archivo no existe.\n");
        Regreso;
    }

    Char Linea[200];
    mientras que (fgets(linea, sizeof(linea), f)) {
        si (parsearEstudiante(linea, &e) && strcmp(e.id, id) == 0) {
            printf("Encontrado: %s %s - Edad %d\n",
                   e.nombres, por ejemplo.Apellidos, E.edad);
            fclose(f);
            Regreso;
        }
    }
    fclose(f);
    printf("ID no encontrado.\n");
}

void actualizarEstudiante() {
    Char ID[20];
    leerCadena("ID a actualizar: ", id, sizeof(id));

    ARCHIVO *f = fopen(ARCHIVO_EST, "r");
    ARCHIVO *temp = fopen("tmp.txt", "w");
    si (!f || ¡Temp) {
        Printf("Error con archivos.\n");
        Regreso;
    }

    Char Linea[200];
    Estudiante e;
    int encontrado = 0;

    mientras que (fgets(linea, sizeof(linea), f)) {
        si (parsearEstudiante(linea, &e) && strcmp(e.id, id) == 0) {
            leerCadena ("Nuevos apellidos: ", e.apellidos, tamaño de(e.apellido
            leerCadena("Nuevos nombres: ", e.nombres, tamaño(e.nombres));
            e.edad  = leerEntero("Nueva edad: ");
            Encontrado = 1;
        }
        fprintf(temp, "%s;%s;%s;%d\n",
                e.id, e.Apellidos, E.nombres, por ejemplo.edad);
    }

    fclose(f);
    Fclose (temp);
    eliminar(ARCHIVO_EST);
    renombrar ("tmp.txt", ARCHIVO_EST);

    si (encontrado)
        printf("Registro actualizado.\n");
    si no,
        printf("ID no encontrado.\n");
}

void eliminarEstudiante() {
    Char ID[20];
    leerCadena("ID a eliminar: ", id, sizeof(id));

    ARCHIVO *f = fopen(ARCHIVO_EST, "r");
    ARCHIVO *temp = fopen("tmp.txt", "w");
    si (!f ||  ¡Temp) {
        printf("Error con archivos.\n");
        Regreso;
    }

    Char Linea[200];
    Estudiante e;
    int eliminado = 0;

    mientras que (fgets(linea, sizeof(linea), f)) {
        si (parsearEstudiante(linea, &e)) {
            si (strcmp(e.id, id) != 0) {
                fprintf(temp, "%s;%s;%s;%d\n",
                        e.id, e.Apellidos, E.nombres, por ejemplo.edad);
            } si no, {
```




Función resultado ← EsPalindro...
Definir invertida Como...
Definir i Como Entero
invertida ← ''
i
Longitu... -1 1
invertida ← invertida+...
resultado ← (cadena=in...
FinFunción


Función salida ← QuitarEspacio...
Definir i Como Entero
salida ← ''
i
1 Longitu...
F
SubCadena(ca...
V
salida ← salida+SubCad...
FinFunción


Función total ← ContarVocales(...
Definir i Como Entero
Definir c Como Cadena
total ← 0
c ← SubCadena(cadena,i,i)
i
1 Longitu...
F
c='a' O c='e...
V
total ← tot
FinFunción


Función encontrado ← BuscarPal...
Definir i Como Entero
encontrado ← Falso
i
1 Longitu...
F
SubCadena(te...
V
encontrado ← Verdadero
FinFunción

https://onlinegdb.com/8X6s1a_LN-