Guía paso a paso: CRUD de Estudiantes con archivos TXT en C (Code::Blocks)

Versión: 1.0 | Fecha: 12/01/2026

FORMATO DE TALLER 1.

DATOS INFORMATIVOS Carrera: Electrónica y automatización

Asignatura: Fundamentos de la Programación

Docente: Jenny Alexandra Ruiz Robalino

Integrantes: Ederson Gualoto, Darwin Tapia, Alex Chuquimarca

Fecha: 14/01/2026

Propósito. Construir un programa en C que permita **Crear, Leer, Actualizar y Eliminar (CRUD)** registros de estudiantes, persistiendo los datos en un fichero de texto (.txt).

Base técnica. La guía se apoya en la presentación "Manejo de Cadenas, Estructuras y Ficheros TXT en C", que explica el uso de **fgets/strcspn** para lectura segura de cadenas, **typedef struct** para modelar un estudiante, y **fopen/fprintf/fgets/sscanf** para escritura/lectura de ficheros TXT, además del patrón de eliminación mediante archivo temporal.

1. Objetivos de aprendizaje

- Definir una estructura Estudiante y capturar datos con lectura segura.
- Guardar estudiantes en un archivo TXT con un formato delimitado (por ejemplo, con ";").
- Listar, buscar, actualizar y eliminar registros reescribiendo el archivo con un fichero temporal.
- Aplicar buenas prácticas: validar fopen != NULL, cerrar ficheros, evitar problemas de buffer y manejar errores.

2. Requisitos y preparación del entorno

- Herramientas sugeridas:
  - Code::Blocks (compilador GCC).
  - Sistema operativo Windows o Linux/macOS (la guía muestra rutas típicas de Windows y una alternativa portable).
- Estructura de carpeta (recomendación):
  - Crear una carpeta del proyecto, por ejemplo: C:\ESTUDIANTE\
  - Dentro, el programa generará/leerá el archivo estudiantes.txt

Nota: En los ejemplos del documento base se usa una ruta absoluta como C:\ESTUDIANTE\estudiantes.txt. Para

simplificar, puedes usar el archivo “estudiantes.txt” en la misma carpeta del ejecutable (ruta relativa).

3. Diseño de datos y formato del archivo TXT

Definimos un registro Estudiante con campos: id, apellidos, nombres y edad (estructura).

- Formato del archivo (una línea por estudiante):
    - id;apellidos;nombres;edad

- Ejemplo:
  - 001;Lucio;Andrea;20

Usaremos ";" como delimitador para poder leer cada línea con fgets() y descomponerla con sscanf().

4. Paso 1 - Definir la estructura y utilidades de lectura segura

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):

```
typedef struct {
    char id[20];
    char apellidos[50];
    char nombres[50];
    int  edad;
} Estudiante;
```

Funciones auxiliares recomendadas:

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

// Elimina el '\n' que deja fgets, si existe.
void limpiarNuevaLinea(char *s) {
    s[strcspn(s, "\n")] = '\0';
}

// Lee una cadena con etiqueta, usando fgets.
void leerCadena(const char *etiqueta, char *dest, int tam) {
    printf("%s", etiqueta);
    fgets(dest, tam, stdin);
    limpiarNuevaLinea(dest);
}

// Lee un entero de forma robusta usando fgets + strtol.
int leerEntero(const char *etiqueta) {
    char buf[50];
    long val;
    char *endptr;

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

        if (endptr != buf) { // hubo conversión
            return (int)val;
```

```
            }
            printf("Entrada inválida. Intente de nuevo.\n");
        }
    }
```

5. Paso 2 - Definir el archivo y el parser de líneas

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.

```
#define ARCHIVO_EST "estudiantes.txt"

// Convierte una línea "id;apellidos;nombres;edad" a estructura Estudiante.
// Retorna 1 si pudo parsear, 0 si la línea es inválida.
int parsearEstudiante(const char *linea, Estudiante *e) {
    // %[^;] = lee hasta encontrar ';'
    int n = sscanf(linea, "%19[^;];%49[^;];%49[^;];%d",
                   e->id, e->apellidos, e->nombres, &e->edad);
    return (n == 4);
}
```

6. Paso 3 - Crear (Agregar) estudiante

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.

1. 6.1. Función para verificar si un ID ya existe:

```
int existeId(const char *idBuscado) {
    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) return 0; // si no existe el archivo, no hay IDs guardados

    char linea[200];
    Estudiante e;

    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e) && strcmp(e.id, idBuscado) == 0) {
            fclose(f);
            return 1;
        }
    }
    fclose(f);
    return 0;
}
```

2. 6.2. Función agregarEstudiante():

```
void agregarEstudiante() {
    Estudiante nuevo;

    leerCadena("Ingrese ID: ", nuevo.id, sizeof(nuevo.id));
```

```
    if (existeId(nuevo.id)) {
        printf("Ya existe un estudiante con ese ID.\n");
        return;
    }

    leerCadena("Ingrese apellidos: ", nuevo.apellidos, sizeof(nuevo.apellidos));
    leerCadena("Ingrese nombres: ",   nuevo.nombres,   sizeof(nuevo.nombres));
    nuevo.edad = leerEntero("Ingrese edad: ");

    FILE *f = fopen(ARCHIVO_EST, "a");
    if (f == NULL) {
        printf("Error: no se pudo abrir el archivo.\n");
        return;
    }

    fprintf(f, "%s;%s;%s;%d\n", nuevo.id, nuevo.apellidos, nuevo.nombres,
nuevo.edad);
    fclose(f);
    printf("Estudiante agregado correctamente.\n");
}
```

7. Paso 4 - Leer (Listar/Consultar) estudiantes

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

```
void listarEstudiantes() {
    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) {
        printf("No hay datos aún (archivo inexistente).\n");
        return;
    }

    char linea[200];
    Estudiante e;

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

    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e)) {
            printf("%-10s %-20s %-20s %-5d\n", e.id, e.apellidos, e.nombres,
e.edad);
        }
    }
    fclose(f);
}
```

8. Paso 5 - Buscar estudiante por ID

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

```
int buscarPorId(const char *idBuscado, Estudiante *salida) {
    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) return 0;

    char linea[200];
    Estudiante e;

    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e) && strcmp(e.id, idBuscado) == 0) {
            *salida = e;
            fclose(f);
            return 1;
        }
    }
    fclose(f);
    return 0;
}

void consultarEstudiante() {
    char id[20];
    Estudiante e;

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

    if (buscarPorId(id, &e)) {
        printf("Encontrado: %s %s (Edad: %d)\n", e.nombres, e.apellidos,
e.edad);
    } else {
        printf("No se encontró el ID.\n");
    }
}
```

9. Paso 6 - Actualizar estudiante (reescritura con archivo temporal)

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().

```
void actualizarEstudiante() {
    char idObjetivo[20];
    leerCadena("Ingrese ID a actualizar: ", idObjetivo, sizeof(idObjetivo));

    FILE *f = fopen(ARCHIVO_EST, "r");
```

```
    if (f == NULL) {
        printf("No existe el archivo.\n");
        return;
    }

    FILE *temp = fopen("tmp.txt", "w");
    if (temp == NULL) {
        fclose(f);
        printf("No se pudo crear archivo temporal.\n");
        return;
    }

    char linea[200];
    Estudiante e;
    int actualizado = 0;

    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e) && strcmp(e.id, idObjetivo) == 0) {
            // Solicitar nuevos datos
            leerCadena("Nuevos apellidos: ", e.apellidos, sizeof(e.apellidos));
            leerCadena("Nuevos nombres: ",   e.nombres,   sizeof(e.nombres));
            e.edad = leerEntero("Nueva edad: ");

            fprintf(temp, "%s;%s;%s;%d\n", e.id, e.apellidos, e.nombres,
e.edad);
            actualizado = 1;
        } else {
            // Copia la línea original (si está bien formateada, mejor re-
escribirla)
            if (parsearEstudiante(linea, &e)) {
                fprintf(temp, "%s;%s;%s;%d\n", e.id, e.apellidos, e.nombres,
e.edad);
            }
        }
    }

    fclose(f);
    fclose(temp);

    remove(ARCHIVO_EST);
    rename("tmp.txt", ARCHIVO_EST);

    if (actualizado) printf("Registro actualizado.\n");
    else printf("No se encontró el ID.\n");
}
```

10. Paso 7 - Eliminar estudiante (baja física)

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.

```
void eliminarEstudiante() {
    char idEliminar[20];
    leerCadena("Ingrese ID a eliminar: ", idEliminar, sizeof(idEliminar));

    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) {
        printf("No existe el archivo.\n");
        return;
    }

    FILE *temp = fopen("tmp.txt", "w");
    if (temp == NULL) {
        fclose(f);
        printf("No se pudo crear archivo temporal.\n");
        return;
    }

    char linea[200];
    Estudiante e;
    int eliminado = 0;

    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e)) {
            if (strcmp(e.id, idEliminar) != 0) {
                fprintf(temp, "%s;%s;%s;%d\n", e.id, e.apellidos, e.nombres,
e.edad);
            } else {
                eliminado = 1;
            }
        }
    }

    fclose(f);
    fclose(temp);

    remove(ARCHIVO_EST);
    rename("tmp.txt", ARCHIVO_EST);

    if (eliminado) printf("Registro eliminado.\n");
    else printf("No se encontró el ID.\n");
}
```

11. Paso 8 - Menú principal (integración CRUD)

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

```
int menu() {
    char opcion[10];
    printf("\n=== CRUD Estudiantes (TXT) ===\n");
    printf("1. Agregar\n");
```

```
    printf("2. Listar\n");
    printf("3. Consultar por ID\n");
    printf("4. Actualizar\n");
    printf("5. Eliminar\n");
    printf("0. Salir\n");
    printf("Seleccione una opción: ");

    fgets(opcion, sizeof(opcion), stdin);
    return atoi(opcion);
}

int main() {
    int op;
    do {
        op = menu();
        switch (op) {
            case 1: agregarEstudiante(); break;
            case 2: listarEstudiantes(); break;
            case 3: consultarEstudiante(); break;
            case 4: actualizarEstudiante(); break;
            case 5: eliminarEstudiante(); break;
            case 0: printf("Saliendo...\n"); break;
            default: printf("Opción inválida.\n");
        }
    } while (op != 0);

    return 0;
}
```

12. Buenas prácticas (para evitar errores comunes)

- Validar siempre fopen(): si devuelve NULL, informar el error y no continuar.
- Cerrar los ficheros con fclose() en todas las rutas de ejecución.
- Preferir fgets() para lectura de texto y eliminar el salto de línea con strcspn().
- Si se usa scanf() para números, limpiar el buffer con getchar() antes de volver a leer cadenas (o mejor, usar fgets + strtol como en esta guía).
- Mantener un formato consistente en el TXT (mismo delimitador y mismo orden de campos).

13. Laboratorio propuesto (entregable práctico)

Objetivo: implementar el CRUD completo y probarlo con al menos 10 registros.

1. Crear el proyecto en Code::Blocks y pegar el código en un solo archivo main.c.

https://onlinegdb.com/3GWQ_98hC

2. Ejecutar y agregar 3 estudiantes. Verificar que se cree estudiantes.txt y que cada estudiante quede en una línea.

```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 2

ID         APELLIDOS            NOMBRES              EDAD
-------------------------------------------------------------
L00457013  Tapia                Darwin               20
L00456713  Chuquimarca          Alex                 20
L00457895  Gualoto              Ederson              20
```

3. 3. Probar la validación de ID duplicado (intentar agregar el mismo ID dos veces).

```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 3
Ingrese ID a buscar: L00456713
Encontrado: Alex Chuquimarca (Edad: 20)

=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 3
Ingrese ID a buscar: L00456083
No se encontr% el ID.
```

4. 4. Listar y verificar el formato de tabla en consola.

```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 4
Ingrese ID a actualizar: L00456713
Nuevos apellidos: Pablo
Nuevos nombres: Jose
Nueva edad: 25
Registro actualizado.

=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 2

ID         APELLIDOS            NOMBRES              EDAD
-------------------------------------------------------------
L00457013  Tapia                Darwin               20
L00456713  Pablo                Jose                 25
L00457895  Gualoto              Ederson              20
```

5. 5. Consultar por ID existente y por uno inexistente.

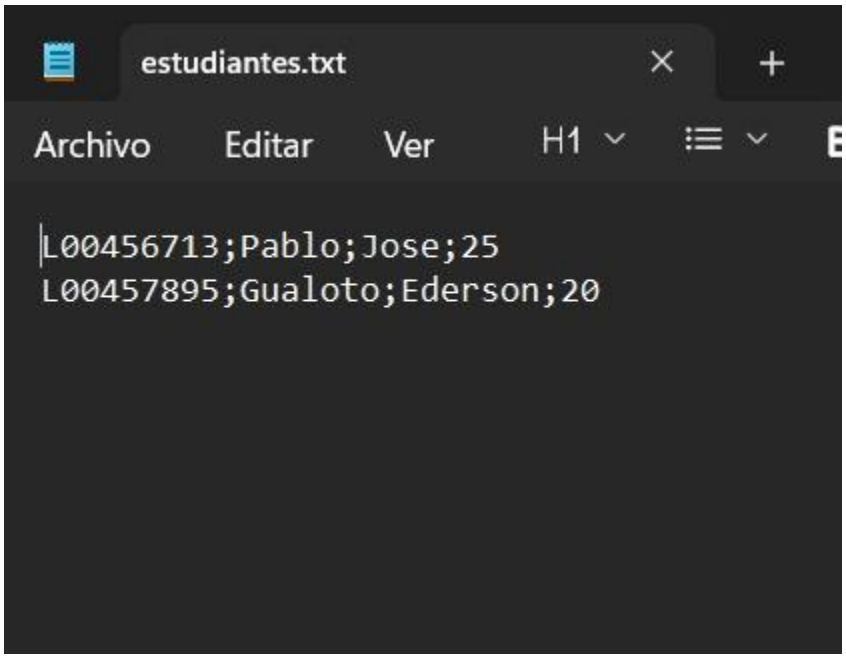
6. 6. Actualizar un estudiante y verificar que los cambios persistan al listar de nuevo.
7. 7. Eliminar un estudiante y verificar que ya no aparece.

```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 5
Ingrese ID a eliminar: L00457013
Registro eliminado.

=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci%n: 2

ID         APELLIDOS            NOMBRES              EDAD
------------------------------------------------------------
L00456713  Pablo                Jose                 25
L00457895  Gualoto              Ederson              20
```

8. 8. Abrir estudiantes.txt con un editor de texto y comprobar que el formato sigue siendo ";" por campos.



Extensión (opcional): Implementar una "baja lógica" agregando un campo estado (A/I) y omitir en listados los registros inactivos.

Referencias

• “Manejo de Cadenas, Estructuras y Ficheros TXT en C”. Presentación utilizada como base para: lectura segura con fgets/strcspn, estructura Estudiante, operaciones con fopen/fprintf/fgets/sscanf y eliminación con archivo temporal (pp. 2-8).

• Joyanes Aguilar, L. (Fundamentos de programación, 4ta ed.). Conceptos de archivo y operaciones sobre registros (altas, bajas y modificaciones) como marco general (capítulo de archivos).