ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELETRÓNICA

Y AUTOMATIZACIÓN

TEMA:

IMPLEMENTACIÓN DE UN CRUD UTILIZANDO ARCHIVOS TXT EN LENGUAJE C

PRESENTADO POR:

OMAR ALQUINGA

INGENIERA:

JENNY RUIZ

FECHA:

13/01/2026

SANGOLQUÍ - ECUADOR

Introducción

Cuando se habla acerca del uso de estructuras en programación, el poder conocer y manejar archivos es parte fundamental para mantener la información de una aplicación, a través de los archivos de texto se puede almacenar o recuperar datos muy fácilmente, esto aplicado en programas simples o pequeños o con fines académicos resulta bastante útil usarlo.

Este informe esta enfocado en el funcionamiento del sistema CRUD (Create, read, update, delete) para hacer un registro de estudiantes mediante el lenguaje de programación C y los archivos de texto con extensión TXT. Haciendo uso de ciertas estructuras, funciones para manejar ficheros, entre otras más funciones y conceptos básicos de programación. En este contexto, (Mendoza Moreno, 2013) define el término CRUD como: “Es el acrónimo de Crear, Obtener (read o retrieve), actualizar (update) y borrar (delete), usado para referirse a las funciones básicas en bases de datos o a la capa de persistencia del software. También se utiliza para describir las convenciones de la interface del usuario que facilitan la vista, búsqueda y el cambio de la información, típicamente utilizado en formas y reportes basados en el computador.”

Desarrollarlo también permiten reforzar las buenas prácticas de programación, como validación de entradas, buen uso de archivos temporales para actualizar y eliminar registros, en resumen, este informe sirve para reforzar y fortalecer lo aprendido en clases y construir una base para la comprensión de sistemas de gestión de datos más difíciles.

Objetivos

Objetivo general:

Desarrollar un programa CRUD para registrar estudiantes haciendo uso del lenguaje de programación tipo C, además de utilizar archivos de texto TXT, estructuras, manejo de cadenas con el fin de aplicar conceptos de manejo de archivos, programación estructurada para tener un programa que funcione correctamente.

Objetivos específicos:

- Registrar y guardas la información de estudiantes con estructuras y funciones para asegurar que se ingrese correctamente los datos y que se almacenen en archivos de texto.
- Leer y consultar información de estudiantes con el uso de los archivos TXT y otras funciones básicas para mostrar los datos y hacer más fácil la búsqueda.
- Actualizar y eliminar los registros de estudiantes con archivos temporales, esto con el fin de mantener la información ordenada.

Marco Teórico

- **Programación estructurada en C**

La programación en C es una de las partes mas importantes en la programación estructurada, porque con el uso de funciones, estructuras de control, uso de funciones se puede desarrollar aplicación o programas útiles. La sintaxis muy fácil de entender y su cercanía al hardware hace que esta sea la herramienta principal a la hora de enseñar principios básicos de programación.

- **Manejo de archivos TXT**

Estos archivos permiten guardar información permanente usando caracteres fáciles de entender y legibles. Para usar estos archivos en lenguaje C se usan los punteros que son tipo FILE y algunas funciones estándar como fprintf, fgets, etc. Funciones como estas hacen mas sencillo la escritura y lectura de datos, ya que hacen que la información se guarde aun después de hacer finalizado la ejecución del programa.

- **Estructuras de datos**

Las estructuras en C (struct) son para agrupar diferentes tipos de datos con un mismo nombre. En este taller la estructura que vamos a utilizar es Estudiante y sirve para mostrar datos importantes de un alumno ya sea su nombre, apellido, edad, etc. Hacer uso de estas estructuras simplifica la manipulación de registros en el programa.

- **Uso de archivos temporales**

Como la forma de leer de los archivos es continua, no se puede modificar o eliminar un registro directamente y aquí es donde entran los archivos temporales, ya que estos copian datos del archivo original pero ya con los cambios requeridos y lo que hace es reemplazar el original por el temporal manteniendo los datos correctos.

Desarrollo

En este taller se realizó un programa con lenguaje de programación tipo C ,el cual permite administrar estudiantes con archivos de texto. Se hizo una estructura que almacenar sus datos, como el ID, nombres, apellidos, edad.

Lo que hace este programa es ingresar estudiantes nuevos, guarda su información en un archivo TXT y comprueba que no haya datos repetidos. Además, muestra los registros guardados y busca a un estudiante por su ID.

En la parte de modificar o eliminar un registro, se hizo uso de archivos temporales, esto se realizó para actualizar la información sin dañar datos existentes, todas estas opciones se encuentran en un menú facilitando el uso del programa.

Código del programa completo en C

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

#define ARCHIVO_EST "estudiantes.txt"

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

```
/* ==========================
   UTILIDADES DE ENTRADA
   ========================== */

// Elimina el '\n' dejado por fgets
void limpiarNuevaLinea(char *s) {
    s[strcspn(s, "\n")] = '\0';
}

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

// Lee un entero de forma robusta
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) {
            return (int)val;
        }
        printf("Entrada inválida. Intente de nuevo.\n");
    }
}

/* ==========================
   PARSER DE LÍNEA
   ========================== */

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

/* ==========================
   VALIDAR ID EXISTENTE
   ========================== */
int existeId(const char *idBuscado) {
    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) {
            fclose(f);
            return 1;
        }
    }
    fclose(f);
    return 0;
}

/* ==========================
   AGREGAR ESTUDIANTE
   ========================== */
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 al 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");
}

/* ==========================
   LISTAR ESTUDIANTES
   ========================== */
void listarEstudiantes() {
    FILE *f = fopen(ARCHIVO_EST, "r");
```

```
    if (f == NULL) {
        printf("No hay datos registrados.\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);
}

/* ==========================
   BUSCAR ESTUDIANTE
   ========================== */
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");
    }
}

/* ==========================
   ACTUALIZAR ESTUDIANTE
   ========================== */
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) {

            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 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");
}

/* ==========================
   ELIMINAR ESTUDIANTE
   ========================== */
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");
}

/* ============================
   MENÚ
   ============================ */
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);
}

/* ============================
   MAIN
   ============================ */
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;
}
```

Pruebas realizadas

- **Registro de múltiples estudiantes:**

```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci¾n: 2

ID         APELLIDOS            NOMBRES              EDAD
-----------------------------------------------------------
L00001     Alquinga             Omar                 19
L00002     Caiza                Katherine            19
L00003     Delgado              Ivanna               18
L00004     Catagña              Andrea               19

=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci¾n:
```

- **Validación de ID duplicado:**

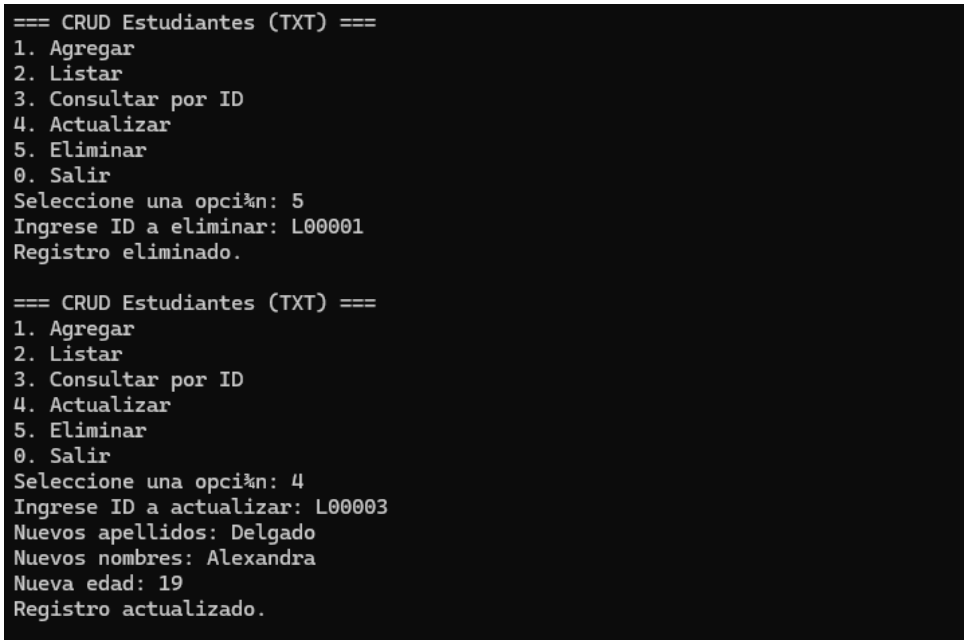
```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci¾n: 1
Ingrese ID: L00001
Ya existe un estudiante con ese ID.
```

- **Consulta por ID existente y no existente:**

```
=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci¾n: 3
Ingrese ID a buscar: L0001
No se encontr¾ el ID.

=== CRUD Estudiantes (TXT) ===
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci¾n: 3
Ingrese ID a buscar: L00003
Encontrado: Ivanna Delgado (Edad: 18)
```

- **Eliminación y actualización de datos:**



Conclusiones

Hacer uso de estructuras, lectura de datos y los archivos temporales hizo mas fácil el organizar información, además de crear, actualizar, y eliminar registros sin ningún tipo de error en el programa.

El haber creado el menú principal facilitó el entender el funcionamiento del programa y la experiencia del usuario permitiendo ejecutar correctamente las funciones de este programa.

Bibliografía

Mendoza Moreno, J. F. (2013). *Generación automática de código de interfaces CRUD, en entornos web a partir de una base de datos para ambientes de software libre.* Obtenido de Repositorio Institucional UNAB: https://repository.unab.edu.co/handle/20.500.12749/3493