RESUMEN

Este documento presenta el desarrollo del Sistema de Gestión de Solicitudes de Mantenimiento, una aplicación en lenguaje C que permite registrar, asignar y dar seguimiento a solicitudes de mantenimiento en una institución. El sistema implementa estructuras de datos dinámicas, gestión de archivos binarios para persistencia y una interfaz de consola interactiva. Desarrollado como proyecto final para la materia de Algoritmos y Estructuras de Datos.

1. OBJETIVOS

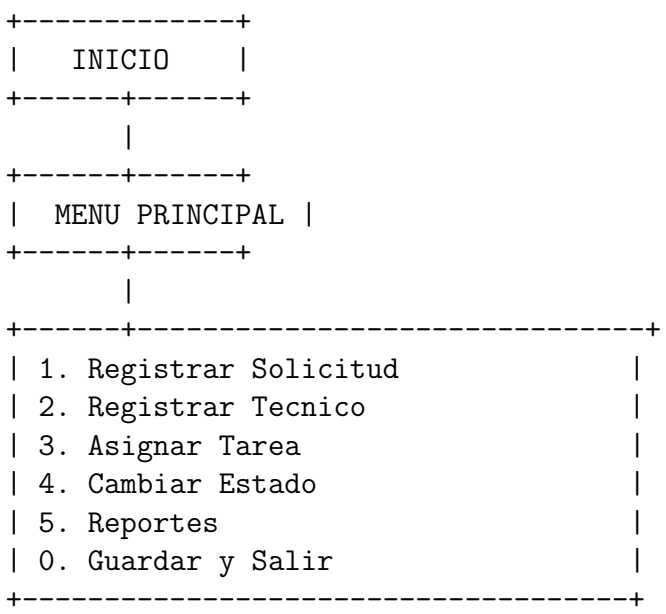
- Modelar procesos administrativos mediante estructuras en C
- Implementar lógica condicional y ciclos para gestión de estados
- Utilizar archivos para persistencia de datos
- Diseñar interfaz de usuario funcional en consola
- Gestionar memoria dinámica de forma eficiente

2. DIAGRAMA DE ESTRUCTURA DEL SISTEMA

2.1. Arquitectura de Archivos

```
.
+-- include/
|   +-- logica.h      - Logica de negocio y estado global
|   +-- persistencia.h - Gestion de archivos binarios
|   +-- tipos.h       - Estructuras y constantes
|   +-- utils.h       - Utilidades de interfaz
+-- src/
|   +-- logica.c      - Implementacion de funciones principales
|   +-- main.c        - Programa principal y menus
|   +-- persistencia.c- Guardado/carga de datos
|   +-- utils.c       - Funciones auxiliares
+-- data/             - Datos persistentes (generado)
|   +-- tecnicos.dat
|   +-- solicitudes.dat
+-- Makefile          - Sistema de compilacion
```

2.2. Diagrama de Flujo del Sistema



3. ESTRUCTURAS DE DATOS

3.1. Definicion de Tipos

```c
// En tipos.h
typedef struct tecnico {
    int id_tecnico;
    char nombre[MAX_NOMBRE];
    char especialidad[MAX_NOMBRE];
    int activo;  // 1=activo, 0=inactivo
} tecnico_t;

typedef struct solicitud {
    int id_solicitud;
    char ubicacion[MAX_UBICACION];
    char descripcion[MAX_DESC];
    int prioridad;           // 1=Baja, 2=Media, 3=Alta
    int estado;              // 1=Pendiente, 2=En Proceso, 3=
   Finalizada
    int id_tecnico_asignado; // ID del tecnico (0 si no asignado)
} solicitud_t;
```

3.2. Variables Globales

```c
// En logica.c
tecnico_t *g_tecnicos = NULL;
```

```c
int g_num_tecnicos = 0;
int g_sig_id_tecnico = 1;

solicitud_t *g_solicitudes = NULL;
int g_num_solicitudes = 0;
int g_sig_id_solicitud = 1;
```

4. DESCRIPCION DE FUNCIONES

4.1. Modulo: logica.c

- **inicializar_datos():** Inicializa punteros globales a NULL/0
- **liberar_memoria():** Libera memoria dinámica de arrays globales
- **registrar_solicitud():** Interfaz para registrar nueva solicitud
- **registrar_tecnico():** Interfaz para registrar nuevo tecnico
- **asignar_tarea():** Asigna solicitud pendiente a tecnico disponible
- **cambiar_estado_solicitud():** Cambia estado de solicitud existente
- **mostrar_reportes():** Submenu para generar reportes diversos

4.2. Modulo: persistencia.c

- **cargar_datos():** Carga datos desde archivos binarios
- **guardar_datos():** Guarda estado actual en archivos binarios
- **asegurar_directorio_de_datos():** Crea directorio data/ si no existe

4.3. Modulo: utils.c

- **limpiar_pantalla():** Limpia consola (multiplataforma)
- **pausar():** Pausa ejecucion hasta Enter
- **leer_opcion():** Lee y valida opcion numerica
- **get_string():** Lee cadena de forma segura

5. FRAGMENTOS CLAVE DEL CODIGO

5.1. Gestion de Memoria Dinamica

```c
void registrar_solicitud(void) {
    g_num_solicitudes++;
    solicitud_t *temp = (solicitud_t *)realloc(
        g_solicitudes,
        g_num_solicitudes * sizeof(solicitud_t)
    );
    if (temp == NULL) {
        perror("Error al reasignar memoria para solicitud");
        g_num_solicitudes--;
        return;
    }
    g_solicitudes = temp;
    // ... resto del codigo
}
```

5.2. Persistencia en Archivos Binarios

```c
void guardar_datos(void) {
    FILE *f;
    asegurar_directorio_de_datos();

    if ((f = fopen(FILE_TECNICOS, "wb")) == NULL) {
        perror("Error al abrir archivo para guardar tecnicos");
        return;
    }
    fwrite(&g_num_tecnicos, sizeof(int), 1, f);
    fwrite(&g_sig_id_tecnico, sizeof(int), 1, f);
    fwrite(g_tecnicos, sizeof(tecnico_t), g_num_tecnicos, f);
    fclose(f);
    // ... similar para solicitudes
}
```

5.3. Logica de Asignacion

```c
void asignar_tarea(void) {
    listar_solicitudes(1); // Solo pendientes
    // ... entrada de datos
    sol->id_tecnico_asignado = id_tec;
    sol->estado = 2; // Cambia a "En Proceso"
    printf("\nAsignacion exitosa: Solicitud %d -> Tecnico %s.\n",
           sol->id_solicitud, tec->nombre);
}
```

6. MANUAL DE USUARIO

6.1. Compilacion y Ejecucion

```
$ make          # Compila el proyecto
$ make clean    # Limpia archivos generados
$ ./bin/gestor_mantenimiento  # Ejecuta el programa
```

6.2. Flujo de Trabajo

1. **Registro Inicial:** Registrar tecnicos antes de asignar solicitudes
2. **Solicitudes:** Cada solicitud se crea con estado "Pendiente"
3. **Asignacion:** Solo solicitudes pendientes pueden asignarse
4. **Seguimiento:** Cambiar estados segun avance el trabajo
5. **Reportes:** Generar reportes por tecnico, ubicacion o estado

6.3. Ejemplo de Uso

```
==================================================
  Sistema de Gestion de Mantenimiento (Proyecto 2)
==================================================
1. Registrar Nueva Solicitud
2. Registrar Nuevo Tecnico
3. Asignar Tarea a Tecnico
4. Actualizar Estado de Solicitud
5. Ver Reportes
0. Guardar y Salir
--------------------------------------------------
Seleccione una opcion (0-5): 1
```

7. PRUEBAS REALIZADAS

7.1. Casos de Prueba

Caso	Entrada	Resultado Esperado	Estado
Registro tecnico	Nombre: "Juan Perez", Especialidad: .Electricidad"	ID asignado automaticamente, tecnico activo	OK
Registro solicitud	Ubicacion: .Oficina 101", Prioridad: Alta	Estado: Pendiente, ID tecnico: 0	OK
Asignacion	IDs validos de solicitud pendiente y tecnico activo	Estado cambia a .En Proceso", tecnico asignado	OK
Cambio estado	Solicitud en proceso - ¿Finalizada	Estado actualizado, tecnico desasignado	OK
Reporte tecnico	ID tecnico con solicitudes asignadas	Lista todas sus solicitudes	OK
Persistencia	Cerrar y reabrir programa	Datos mantienen estado anterior	OK

8. CONCLUSIONES Y MEJORAS FUTURAS

8.1. Conclusiones

- El sistema cumple con todos los requisitos funcionales especificados
- La arquitectura modular facilita el mantenimiento y extension
- El uso de memoria dinámica permite escalabilidad
- La persistencia en archivos binarios garantiza integridad de datos
- La interfaz de consola es intuitiva y robusta ante entradas invalidas

8.2. Mejoras Futuras

1. **Interfaz Grafica:** Implementar version con GTK o Qt
2. **Base de Datos:** Migrar de archivos binarios a SQLite
3. **Autenticacion:** Sistema de usuarios y roles
4. **Notificaciones:** Integracion con correo electronico

5. **Metricas:** Estadisticas avanzadas y graficos

6. **Backup:** Sistema automatico de respaldo de datos

9. BIBLIOGRAFIA

1. Kernighan, B. W. & Ritchie, D. M. (2019). *El lenguaje de programación C* (2.a ed.). Pearson Educación.

2. García, M., Rodríguez, P. & López, J. (2021). *Estructuras de datos en C: Implementación y aplicaciones*. McGraw-Hill Interamericana.

3. Silva, A. (2020). *Programación en C: Desde fundamentos hasta aplicaciones avanzadas*. Editorial Tébar.

4. Organización Internacional para la Estandarización. (2018). *ISO/IEC 9899:2018: Lenguajes de programación — C*. ISO.

5. Martínez, C. & Fernández, R. (2019). *Algoritmos y estructuras de datos en C*. Ediciones Paraninfo.

6. González, L. (2022). *Desarrollo de sistemas en lenguaje C: Buenas prácticas y patrones de diseño*. Ra-Ma Editorial.