DOCUMENTACIÓN:

Descripción General:

Este simulador básico, permite gestionar tareas dividiéndolas en pendientes y completadas. La clave de su estructura radica en dos tipos distintos de listas enlazadas que modelan comportamientos distintos:

1. Lista enlazada ordenada por prioridad: Para las tareas pendientes
2. Cola Enlazada (FIFO): Para las tareas completadas

Además, se aplican estrategias algorítmicas fundamentales para simular operaciones como inserción ordenada, búsqueda por descripción y eliminación de un nodo específico.

Estructuras de Datos:

1. Lista Enlazada ordenada por prioridad (Tareas pendientes):

-Se usa para insertar tareas manteniendo el orden por prioridad de forma descendente.

```
typedef struct Tarea {
    char descripcion[MAX_DESC];
    int prioridad;
    struct Tarea *sig;
} Tarea;
```

-Usada en:

- agregarTarea()
- listarPendientes()
- buscarTarea()

- MarcarCompletadaPorIndice()
- GuardarTareas()
- CargarTareas()

-Algoritmo aplicado:

- Inserción en lista ordenada por prioridad descendente.
    - Recorre la lista hasta encontrar el primer nodo cuya prioridad sea menor o igual que la nueva
    - Inserta antes para mantener orden

2. Cola Enlazada FIFO (Tareas completadas):

- Se comporta como una cola simple, donde las tareas completadas se agregan al final y se muestran en orden de finalización:

```
typedef struct NodoCompletada {
    char descripcion[MAX_DESC];
    int prioridad;
    struct NodoCompletada *sig;
} NodoCompletada;
```

-Usada en:

- MarcarCompletadaPorIndice()
- MostrarCompletadas()
- BuscarTarea()
- GuardarTareas()
- CargarTareas()

-Algoritmo aplicado:

- Inserción al final de la cola

- Recorre la cola hasta encontrar el último nodo (sig == NULL) y agrega al final

Funciones y Estrategias Algorítmicas:

1. agregarTarea (char *desc, int prioridad):

-Inserta en la lista de pendientes

-Algoritmo: Inserción Ordenada en lista enlazada

- Casos tratados:
  - Lista Vacía
  - Nueva tarea con prioridad mayor (se convierte en head)
  - Inserción en medio o al final

2. buscarTarea(char *desc):

-Búsqueda secuencial en ambas listas enlazadas (pendientes y completadas)

-Algoritmo: Comparación parcial (strstr) para encontrar descripciones que contengan el texto buscado

3. marcarCompletadaPorIndice(int índice):

-Eliminación por posición en lista enlazada (no por contenido)

- Recorre con contador hasta encontrar el índice deseado

-Guarda los datos antes de liberar el nodo

-Inserta la tarea eliminada al final de la cola de tareas completadas

4. guardarTareas() y cargarTareas():

-Serialización y deserialización de ambas listas enlazadas en archivos separados.

-Usan el formato prioridad|descripcion.

5. main()

-Contiene un **menú interactivo** y control del flujo. Ejecuta la carga de tareas al inicio y guarda al final.

Tabla resumen: estructuras, funciones:

FUNCIÓN	TIPO DE LISTA	ESTRATEGIA IMPLICADA
agregarTarea()	Lista ordenada (pend.)	Inserción ordenada por prioridad.
listarPendientes()	Lista ordenada (pend.	Recorrido Simple
marcarCompletadaPorIndice()	Lista ordenada + Cola	Eliminación + inserción al final
mostrarCompletadas()	Cola	Recorrido FIFO
buscarTarea()	Ambas	Búsqueda secuencial por coincidencia
guardarTareas()	Ambas	Serialización
cargarTareas()	Ambas	Reconstrucción de estructuras

Consideraciones Importantes:

- Gestión de memoria: Cada tarea se crea dinámicamente con malloc() y se libera correctamente con free().
- Separación de listas: Las tareas pendientes y completadas son independientes en estructura y lógica.
- Orden y prioridad: La lista de pendientes siempre refleja correctamente el orden de ejecución preferente.
- **EXTRA**: Persistencia, Se asegura la continuidad del estado de ejecución entre sesiones mediante el manejo de archivos.txt