

El código en C que presentas hace uso intensivo de **apuntadores (punteros)**, especialmente para:

1. **Manipular datos de entrada y salida,**
2. **Trabajar con arreglos y cadenas, y**
3. **Pasar variables por referencia para permitir que funciones las modifiquen.**

A continuación te explico **detalladamente todos los usos de apuntadores** en tu código:



1. En la función `cargarAviones`

```
int cargarAviones(Avion aviones[], const char *nombreArchivo)
```

Apuntador: `const char *nombreArchivo`

- Es un **puntero a una cadena constante**.
- Se utiliza para recibir el **nombre del archivo** como parámetro, por ejemplo `"aviones.txt"`.
- El `const` indica que no se modificará el contenido de esa cadena dentro de la función.

Apuntador: `FILE *archivo`

- `fopen` retorna un puntero a un objeto `FILE`, que representa el archivo abierto.
- Usas `archivo` para leer del archivo.

Apuntadores en `fscanf`:

```
fscanf(archivo, ..., aviones[i].nombre, &aviones[i].capacidad, ...)
```

- `&aviones[i].capacidad, &aviones[i].alcance, etc.:`
Se pasan direcciones de memoria para que `fscanf` pueda escribir directamente en esas variables.
Es un uso **esencial de punteros para entrada de datos**.
-



2. En la función `seleccionarOpcion`

```
void seleccionarOpcion(const char *mensaje, const char *opciones[], int cantidad, int *resultado)
```

Apuntador: `const char *mensaje`

- Puntero a una cadena constante (texto a mostrar como mensaje).

Apuntador: `const char *opciones[]`

- Arreglo de punteros a cadenas constantes.
- Cada `opciones[i]` apunta a un string como `"Comercial"` o `"Privado"`.
- Este patrón es típico para manejar listas de texto dinámicamente.

Apuntador: `int *resultado`

- Se usa para **pasar un entero por referencia**.
 - La función escribe el índice elegido por el usuario en la dirección `resultado`.
 - Ejemplo de uso en `main`:
 - `seleccionarOpcion(..., &tipo_index);`
-

3. En la función `pedirParametrosBusqueda`

```
void pedirParametrosBusqueda(int *capacidadMin, int *alcanceMin, float *precioMax, float *precioHoraMax)
```

- Todos los parámetros son punteros.
 - El objetivo es que la función pueda **modificar directamente** los valores de estas variables, usando `scanf`.
 - Ejemplo:
 - `scanf("%i", capacidadMin); // capacidadMin es un puntero a int`
-

4. En la función `cumpleCriterios`

```
int cumpleCriterios(Avion a, const char *tipo, const char *motor, const char *ala, const char *categoria, ...)
```

Apuntadores: `const char *tipo, motor, ala, categoria`

- Cada uno es un puntero a cadena constante.
- Se usan para comparar texto con `strcmp`.

```
(tipo == NULL || strcmp(a.tipo, tipo) == 0)
```

- Si `tipo == NULL` (caso "Cualquiera"), no se filtra por tipo.
- Si no es `NULL`, se compara el string con el campo `a.tipo`.

Esto muestra cómo los punteros permiten flexibilidad al pasar “filtros opcionales”.

5. En la función `main`

Arreglos de punteros a cadenas

```
const char *tipos[] = {"Cualquiera", "Comercial", "Privado", "Militar"};
```

- Cada elemento de `tipos` es un **puntero a una cadena**.
- Arreglo de punteros: puedes pasar fácilmente listas de strings a funciones como `seleccionarOpcion`.

Asignación condicional de punteros:

```
const char *tipoFiltro = (tipo_index == 0) ? NULL : tipos[tipo_index];
```

- Si el usuario elige "Cualquiera", se asigna `NULL` (sin filtro).
- Si no, se asigna el puntero correspondiente al string.

Esto se pasa luego a `cumpleCriterios`.

6. Apuntadores implícitos en cadenas de caracteres

Cada vez que haces algo como:

```
strcmp(a.tipo, tipo)
```

Estás trabajando con punteros. Las cadenas de caracteres (`char[]` o `char *`) en C son en realidad **arreglos terminados en NULL**, y cuando se pasan a funciones, se pasan como punteros.

¿Por qué se usan punteros aquí?


1. **Evitan copiar grandes estructuras o arreglos** innecesariamente (como arreglos de strings).
2. **Permiten modificar variables** desde funciones (paso por referencia).
3. **Facilitan el manejo de texto y archivos**, que en C siempre requiere punteros.

Resumen por tipo de uso

Tipo de puntero	Uso principal	Ejemplo
<code>char *</code> o <code>const char *</code>	Manejar texto (mensajes, cadenas, filtros)	<code>const char *mensaje</code>
<code>char *opciones[]</code>	Arreglo de punteros a cadenas	<code>const char *tipos[]</code>
<code>int *</code> / <code>float *</code>	Paso por referencia para modificar valores	<code>scanf("%i", capacidadMin);</code>
<code>FILE *</code>	Archivo como flujo de datos	<code>FILE *archivo = fopen(...);</code>
<code>NULL</code> como valor especial de puntero	Indicar ausencia de filtro o valor opcional	<code>tipoFiltro = (tipo_index == 0) ? NULL : ...</code>

Si necesitas una versión visual o un ejemplo paso a paso con diagramas de memoria, también te puedo ayudar con eso.

Todo lo siguiente son Ejemplos mas explicados de cada función:

 `pedirParametrosBusqueda(int *capacidadMin, int *alcanceMin, float *precioMax, float *precioHoraMax)`

Esta función **modifica múltiples variables de tipo int y float en main** utilizando punteros. Vamos a ilustrarlo con un ejemplo completo.

Fragmento relevante en main:

```
int capacidadMin, alcanceMin;
float precioMax, precioHoraMax;

pedirParametrosBusqueda(&capacidadMin, &alcanceMin, &precioMax, &precioHoraMax);
```

Paso 1: Antes de la llamada

```
int capacidadMin, alcanceMin;
float precioMax, precioHoraMax;
```

En memoria:

+-----+	+-----+
capacidadMin: ?	alcanceMin: ?
+-----+	+-----+
+-----+	+-----+
precioMax: ?	precioHoraMax: ?
+-----+	+-----+

Paso 2: Llamada a la función

```
pedirParametrosBusqueda(&capacidadMin, &alcanceMin, &precioMax, &precioHoraMax);
```

La función recibe **punteros** a estas variables:

- `capacidadMin` → dirección de memoria: `&capacidadMin`
 - `alcanceMin` → `&alcanceMin`
 - `precioMax` → `&precioMax`
 - `precioHoraMax` → `&precioHoraMax`
-

Paso 3: Dentro de la función

```

void pedirParametrosBusqueda(int *capacidadMin, int *alcanceMin, float *precioMax, float
*precioHoraMax) {
    printf("Capacidad minima: ");
    scanf("%i", capacidadMin);

    printf("Alcance minimo (km): ");
    scanf("%i", alcanceMin);

    printf("Precio maximo (millones USD): ");
    scanf("%f", precioMax);

    printf("Precio maximo por hora de vuelo (USD): ");
    scanf("%f", precioHoraMax);
}

```

Supongamos que el usuario escribe:

- Capacidad mínima: 150
- Alcance mínimo: 8000
- Precio máximo: 120.5
- Precio por hora: 3000.0

📌 Paso 4: Visualización de `scanf` con punteros

Línea	Qué pasa (internamente)
<code>scanf("%i", capacidadMin)</code>	Escribe en la dirección <code>&capacidadMin</code>
<code>scanf("%f", precioMax)</code>	Escribe en la dirección <code>&precioMax</code>

📌 Estado de la memoria después:

+-----+	+-----+
capacidadMin:150	alcanceMin:8000
+-----+	+-----+
+-----+	+-----+
precioMax:120.5	precioHoraMax:3000.0
+-----+	+-----+

📌 ¿Qué hace el puntero en cada caso?

Parámetro	Tipo	Qué contiene
<code>int *capacidadMin</code>	Puntero a <code>int</code>	Dirección de <code>capacidadMin</code> en <code>main</code>
<code>float *precioMax</code>	Puntero a <code>float</code>	Dirección de <code>precioMax</code> en <code>main</code>

Así, al hacer `scanf("%f", precioMax);`, se escribe directamente en la variable original de `main`.

Conclusión

- `scanf("%i", ptr)` necesita un **puntero** para saber **dónde escribir el valor leído**.
 - La función `pedirParametrosBusqueda` recibe punteros y escribe directamente en las variables originales de `main`.
 - Esto permite que múltiples valores sean modificados y "retornados" sin usar `return`.
-

¿Te gustaría que ahora veamos una visualización con una función que **recibe punteros a estructuras o devuelve estructuras**? ¿O prefieres seguir con `cumpleCriterios` y cómo usa punteros a cadenas (`const char *`)?

Parte del código que vamos a ilustrar:

```
const char *tipos[] = {"Cualquiera", "Comercial", "Privado", "Militar"};
int tipo_index;

seleccionarOpcion("Seleccione el tipo de avion:", tipos, 4, &tipo_index);
```

Y la función:

```
void seleccionarOpcion(const char *mensaje, const char *opciones[], int cantidad, int
*resultado) {
    int opcion = -1;
    while (opcion < 1 || opcion > cantidad) {
        printf("%s\n", mensaje);
        for (int i = 0; i < cantidad; i++) {
            printf("%i. %s\n", i + 1, opciones[i]);
        }
        printf("Selecciona una opcion: ");
        scanf("%i", &opcion);
        if (opcion < 1 || opcion > cantidad) {
            printf("Opcion invalida. Intenta de nuevo.\n\n");
        }
    }
    *resultado = opcion - 1;
}
```

 **Objetivo: entender cómo los punteros trabajan para devolver el índice seleccionado.**

Diagrama de memoria y flujo visual

1. Antes de llamar a la función

```
const char *tipos[] = {"Cualquiera", "Comercial", "Privado", "Militar"};
int tipo_index;
```

Memoria:

```
tipos[0] → "Cualquiera"
tipos[1] → "Comercial"
tipos[2] → "Privado"
tipos[3] → "Militar"
```

```
tipo_index: (sin inicializar aún)
```

2. Llamada a la función

```
seleccionarOpcion("Seleccione el tipo de avion:", tipos, 4, &tipo_index);
```

Parámetros que recibe la función:

- mensaje → puntero a "Seleccione el tipo de avion:"
- opciones → mismo arreglo de punteros que tipos
- cantidad → 4
- resultado → puntero a `tipo_index` en memoria

Visualmente:

resultado → tipo_index (en memoria)

```
+-----+
| tipo_index: ??? |
+-----+
```

3. Dentro de la función (ejemplo con entrada del usuario = 2)

```
scanf("%i", &opcion);
*resultado = opcion - 1;
```

Supongamos que el usuario elige la opción **2** ("Comercial"). Entonces:

- `opcion = 2`
- `*resultado = 2 - 1 = 1`
- Como resultado apunta a `tipo_index`, se modifica su valor.

Resultado:

```
tipo_index = 1
```

Memoria después:

```
+-----+
| tipo_index: 1 |
+-----+
```

4. De vuelta en main

Ahora que `tipo_index == 1`, se hace:

```
const char *tipoFiltro = (tipo_index == 0) ? NULL : tipos[tipo_index];
```

```
→ tipoFiltro = tipos[1] → "Comercial"
```

✿ Relación entre punteros y lo que ocurre

Elemento	Qué es	Qué hace el puntero
<code>const char *</code>	Puntero a cadena	Apunta a una cadena de texto
<code>char *opciones[]</code>	Arreglo de punteros a cadenas	Cada <code>opciones[i]</code> apunta a un string
<code>int *resultado</code>	Puntero a <code>tipo_index</code>	Permite que la función lo modifique

Elemento	Qué es	Qué hace el puntero
<code>*resultado = val</code>	Desreferencia	Escribe en la variable apuntada



Conclusión visual

- Los punteros te permiten **acceder y modificar valores** fuera del alcance local de una función.
 - En este ejemplo, gracias al puntero `resultado`, la función `seleccionarOpcion` modifica directamente `tipo_index` definido en `main`.
 - El arreglo de cadenas `tipos[]` es en realidad un arreglo de **punteros a texto**.
-

Definición de la función `cumpleCriterios`

```
int cumpleCriterios(Avion a, const char *tipo, const char *motor, const char *ala, const char *categoria,
                    int capacidadMin, int alcanceMin, float precioMax, float
precioHoraMax) {
    return
        (tipo == NULL || strcmp(a.tipo, tipo) == 0) &&
        (motor == NULL || strcmp(a.motor, motor) == 0) &&
        (ala == NULL || strcmp(a.ala, ala) == 0) &&
        (categoria == NULL || strcmp(a.categoria, categoria) == 0) &&
        a.capacidad >= capacidadMin &&
        a.alcance >= alcanceMin &&
        a.precio <= precioMax &&
        a.precioHora <= precioHoraMax;
}
```

¿Qué hace esta función?

Comprueba si un **avión a** cumple con los filtros indicados:

- Si alguno de los filtros (`tipo`, `motor`, `ala`, `categoria`) es `NULL`, se ignora.
 - Compara cadenas con `strcmp(...) == 0` si el filtro no es `NULL`.
-

En `main`:

```
const char *tipoFiltro = (tipo_index == 0) ? NULL : tipos[tipo_index];
```

Y luego se llama:

```
cumpleCriterios(aviones[i], tipoFiltro, motorFiltro, alaFiltro, categoriaFiltro,
                capacidadMin, alcanceMin, precioMax, precioHoraMax);
```

Visualización en memoria

Supongamos que:

```
const char *tipoFiltro = "Comercial";
const char *motorFiltro = NULL; // El usuario seleccionó "Cualquiera"
...
Avion a = aviones[i]; // Supongamos un avión llamado "Boeing 737"
```


Y que:

```
a.tipo = "Comercial"
a.motor = "Turbofan"
a.ala = "Fija"
```

```
a.categoria = "Servicio"
a.capacidad = 180
a.alcance = 4000
a.precio = 90.0
a.precioHora = 2500.0
```

📌 Evaluación de condiciones

```
(tipo == NULL || strcmp(a.tipo, tipo) == 0)
```

- `tipo != NULL` → "Comercial"
- `a.tipo = "Comercial"`
- → `strcmp("Comercial", "Comercial") == 0` → 

```
(motor == NULL || strcmp(a.motor, motor) == 0)
```

- `motor == NULL` →  se ignora este filtro
-

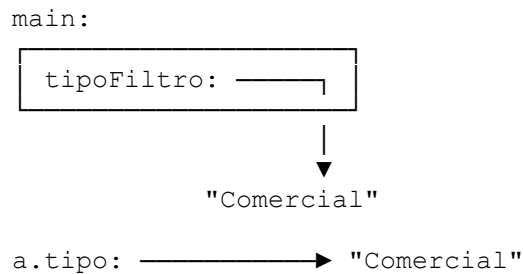
📌 ¿Qué son `const char *`?

Son **punteros a cadenas de texto**. Por ejemplo:

```
tipoFiltro → "Comercial"
a.tipo      → "Comercial"
```

La comparación se hace con `strcmp`, que compara el contenido de esas cadenas, no las direcciones.

🖼 Diagrama visual:



Entonces:

```
strcmp(a.tipo, tipoFiltro) == 0 → true
```

📌 ¿Qué pasa si `tipoFiltro` es `NULL`?

```
(tipo == NULL || strcmp(a.tipo, tipo) == 0)
```

- Si `tipoFiltro == NULL`, entonces el resultado de esta parte será automáticamente `true`.
- Esto permite que el usuario diga: "no me importa el tipo", y que no se filtre por él.

Conclusión

Elemento	Tipo	Propósito
<code>const char *tipo</code>	puntero a cadena	Filtro opcional, puede ser <code>NULL</code> o una cadena
<code>a.tipo</code>	cadena dentro de struct	Se compara con el filtro usando <code>strcmp()</code>
<code>strcmp()</code>	compara strings	Devuelve 0 si las cadenas son iguales
<code>tipo == NULL</code>	ignora el filtro	Útil para aplicar o no aplicar condiciones
