

# PUNTEROS EN C

*Universidad Tecnológica Nacional - Cuch Sede Chivilcoy*

# Concepto

Un puntero es una variable que almacena la dirección de memoria de otra variable. En lugar de contener un valor directamente, contiene la ubicación de donde se encuentra almacenado el valor en la memoria

```
int *ptr;
```

ptr es un puntero a un entero. Esto significa que ptr puede almacenar la dirección de memoria de una variable entera.

```
int x = 10;
```

```
ptr = &x;
```



# Concepto

La variable ptr entonces contiene la dirección de memoria de x. Para acceder al valor de x a través de ptr, se utiliza el **operador de indirección \***:

```
printf("%d", *ptr); // Esto va a imprimir el valor de x, que es 10
```



# ¿Por qué son importantes?

Los punteros son fundamentales en C por varias razones:

Permiten acceso directo a la memoria

Facilitan el manejo eficiente de estructuras de datos

Posibilitan la asignación dinámica de memoria

Permiten modificar variables dentro de funciones (paso por referencia)

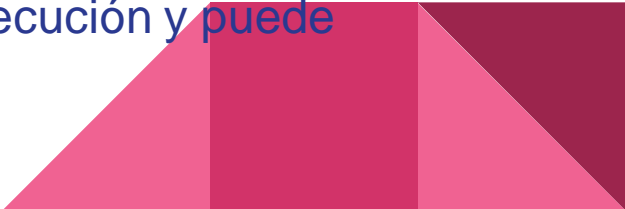
Son la base para trabajar con arrays y cadenas de caracteres



# Usos

**Gestión de la memoria dinámica:** Los punteros son esenciales para asignar y liberar memoria dinámicamente utilizando funciones como **malloc()**, **calloc()**, **realloc()** y **free()**. Esto es útil cuando no se conoce el tamaño de los datos en tiempo de compilación o cuando se necesita asignar y liberar memoria según sea necesario durante la ejecución del programa.

La diferencia fundamental entre asignar memoria estáticamente y dinámicamente es que, en el primer caso, la memoria se asigna en tiempo de compilación y no se puede cambiar en tiempo de ejecución, mientras que en el segundo caso, la memoria se asigna en tiempo de ejecución y puede cambiarse dinámicamente según sea necesario.



# Usos

**Manipulación de cadenas de caracteres:** Las cadenas de caracteres en C se representan como matrices de caracteres terminadas con el carácter nulo ('\0'). Los punteros se utilizan para acceder, manipular y recorrer los elementos de estas cadenas de caracteres de manera eficiente.



# Usos

**Paso de argumentos a funciones por referencia:** Los punteros permiten pasar argumentos a funciones por referencia en lugar de por valor. Esto significa que la función puede modificar directamente los datos en la memoria principal en lugar de hacer copias de los mismos. Esto puede ser útil para conservar recursos y para manipular grandes cantidades de datos de manera eficiente.



# Usos

**Implementación de estructuras de datos avanzadas:** Los punteros son esenciales para implementar estructuras de datos dinámicas como listas enlazadas, pilas, colas, árboles y grafos. Estas estructuras de datos a menudo requieren que los nodos o elementos se vinculen entre sí mediante punteros.





# Los Arrays siempre pasan por referencia

- **Eficiencia de memoria:** Los arrays pueden ser grandes y ocupar una cantidad significativa de memoria. Pasarlos por valor (copiando todo el contenido del array) sería ineficiente, especialmente en términos de uso de memoria, ya que requeriría duplicar la cantidad de memoria necesaria.
  - **Tiempo de ejecución:** Copiar un array grande también llevaría tiempo, lo que afectaría el rendimiento del programa.
  - **Consistencia de punteros:** En C, el nombre de un array actúa como un puntero al primer elemento del array. Pasar un array a una función simplemente pasa la dirección de memoria del primer elemento. Esto es consistente con la idea de pasar punteros a funciones, lo que significa que las funciones pueden manipular directamente los datos en el array original.
  - **Necesidad de cambios:** Si una función modifica un array, estos cambios se reflejan en el original.
  - **Arrays de tamaño variable:** Los arrays en C **no** pueden tener un tamaño variable. Pasarlos por referencia permite que una función trabaje con un array de cualquier tamaño sin la necesidad de conocer su tamaño en tiempo de compilación.
- 