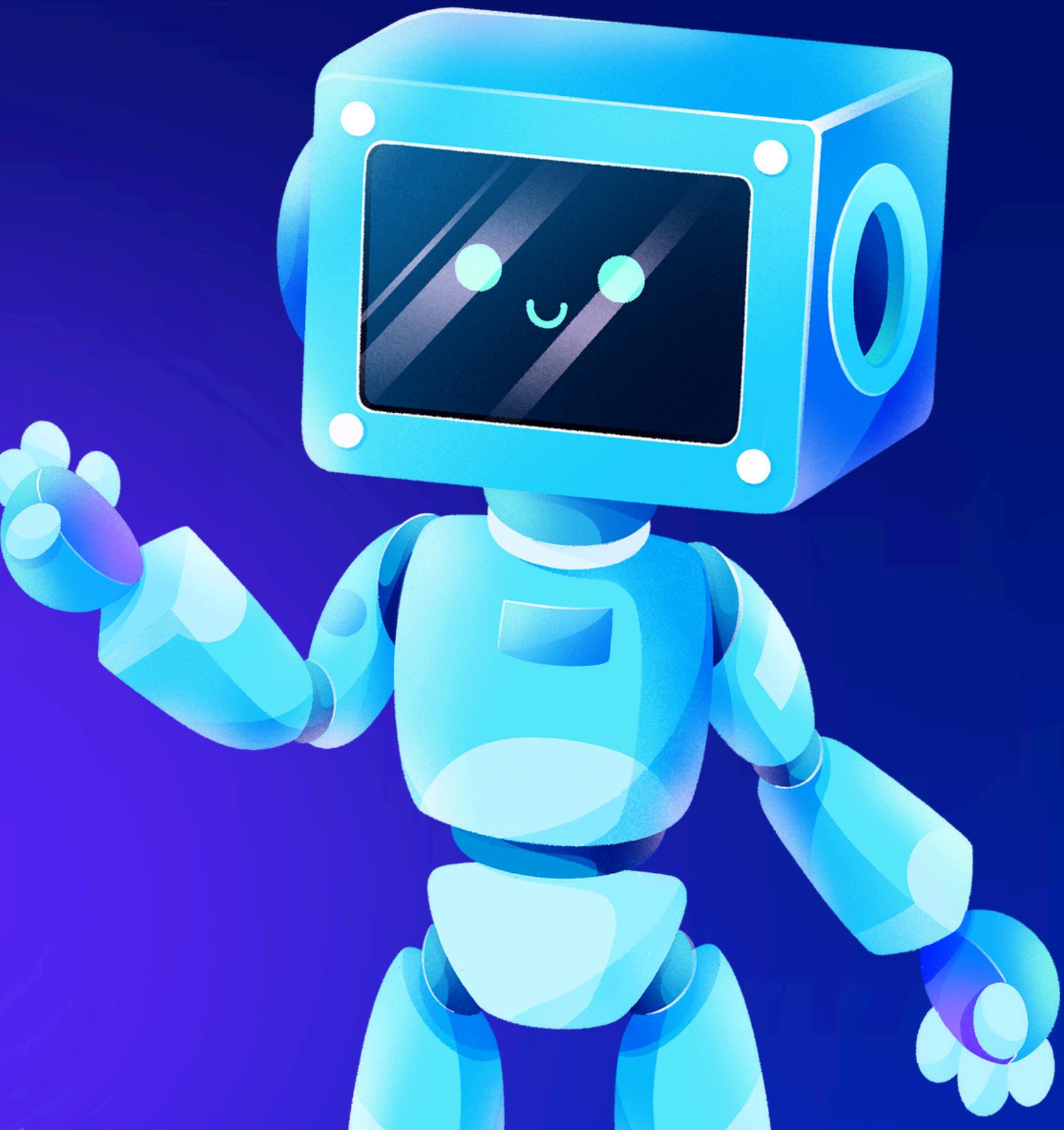




TAREA AUSENTE

PUNEROS

by Jhonatan

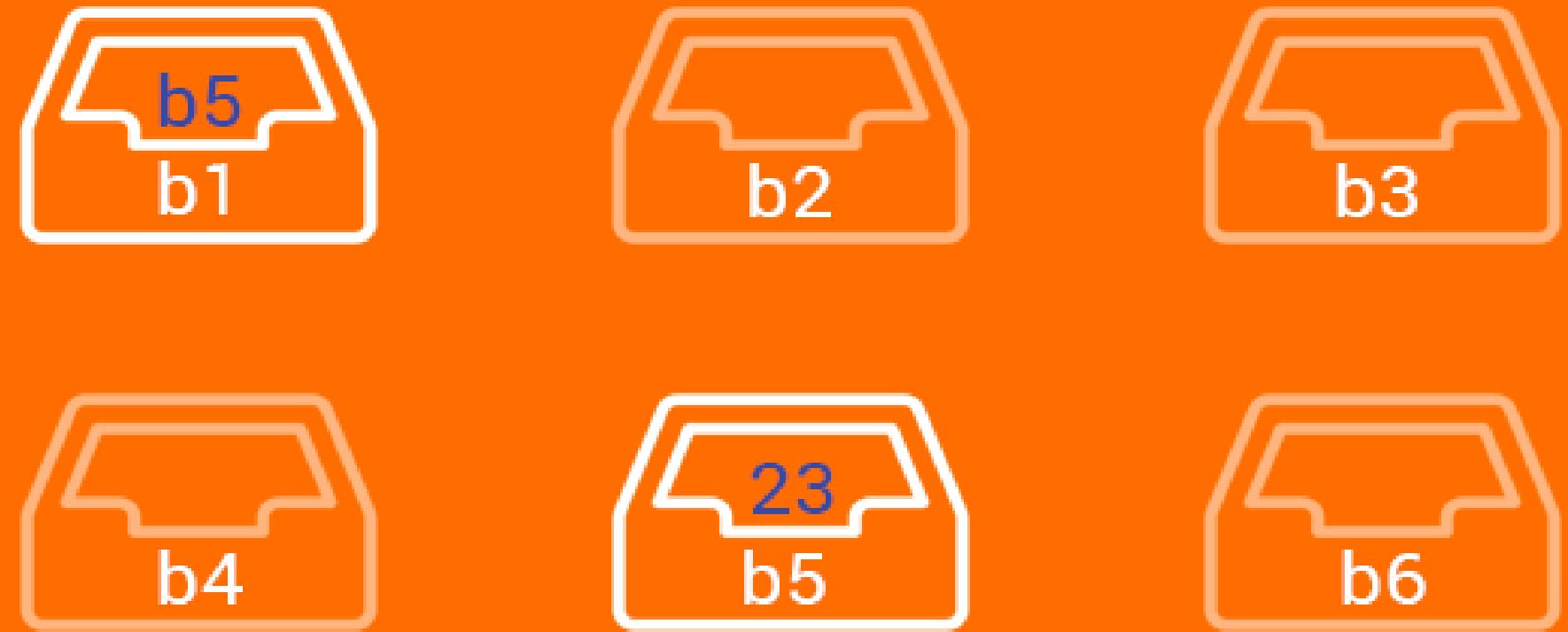


PUNTEROS (C++)

En la programación de estilo C, se usan punteros básicos para todos estos escenarios. Sin embargo, los punteros básicos son el origen de muchos errores de programación graves. Por lo tanto, se desaconseja encarecidamente su uso, excepto cuando proporcionan una ventaja significativa de rendimiento y no hay ambigüedad en cuanto a qué puntero es el puntero propietario que es responsable de eliminar el objeto. El lenguaje C++ moderno proporciona punteros inteligentes para asignar objetos, iteradores para recorrer estructuras de datos y expresiones lambda para pasar funciones. Al usar estas características del lenguaje y de la biblioteca en lugar de punteros básicos, hará que su programa sea más seguro, más fácil de depurar y más fácil de entende

CONCEPTO

Un puntero es una variable que contiene la dirección de memoria de otra variable que contiene al dato en un arreglo. Esto quiere decir, que el puntero apunta al espacio físico donde está el dato o la variable. Un puntero puede apuntar a un objeto de cualquier tipo, como por ejemplo, a una estructura o una función. Los punteros se pueden utilizar para referencia y manipular estructuras de datos, para referenciar bloques de memoria asignados dinámicamente y para proveer el paso de argumentos por referencias en las llamadas a funciones. Muchas de las funciones estándares de C, trabajan con punteros, como es el caso del scanf o strcpy. Estas funciones reciben o devuelven un valor que es un puntero. Por ejemplo: A scanf se le pasa la dirección de memoria del dato a leer...



Memoria

Dirección	Contenido	Tipo de dato
0x67	5	int
0x75	0x67	int*
0x88	0x75	int**

CONCEPTOS

OPERADORES ASOCIADOS A PUNTEROS

¿Qué es el puntero en C?

La puntero en C, es una variable que almacena la dirección de otra variable. También se puede utilizar un puntero para hacer referencia a otra función de puntero.

Un puntero se puede incrementar/dismuir, es decir, apuntar a la ubicación de memoria siguiente/anterior. El propósito del puntero es ahorrar espacio en la memoria y lograr un tiempo de ejecución más rápido.

01

Si declaramos
una variable v
de tipo int, v
realmente
almacenará un
valor.

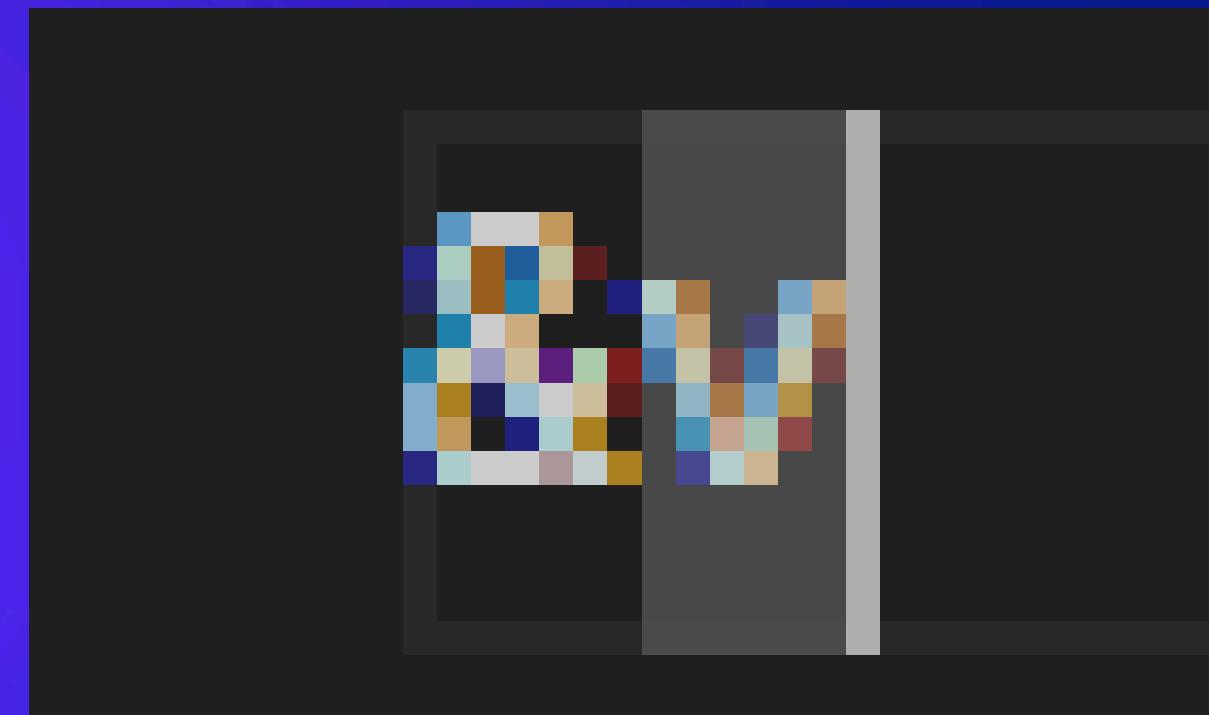
```
int v = 0;
```

EJEMPLOS:

02

v es igual a cero ahora.

Sin embargo, cada variable, además del valor,
también tiene su dirección (o, simplemente,
dónde se encuentra en la memoria). La
dirección se puede recuperar poniendo un signo
(&) antes del nombre de la variable.





03

```
#include <stdio.h>

int main() {
    int v = 0;
    printf("%d\n", &v);
    return 0;
}
```

Si imprimes la dirección de una variable en pantalla, se verá como un número totalmente aleatorio (además, puede ser diferente de una ejecución a otra).

Probemos esto en la práctica con el ejemplo de puntero en C.

TIPOS DE DATOS DE PUNTEROS

1. Tipos de punteros en C:

- Puntero a entero (int*): Almacena la dirección de una variable de tipo entero.
- Puntero a doble (double*): Apunta a una variable de tipo doble (número decimal).
- Puntero a flotante (float*): Se utiliza para variables de tipo flotante (números con decimales).
- Puntero a carácter (char*): Apunta a una variable de tipo carácter.

¿COMO SE DECLARAN Y SE LLAMAN?

para declarar una variable/valor con un puntero en c es como lo hacemos normalmente es decir que pones el tipo de dato por ejemplo int y despues el nombre de nuestra variable solo que ahora le agregamos un asterisco al comienzo del nombre de nuestra variable En la siguiente hoja ejemplo: como tambien a la hora de llamarlo es como siempre solo que agregamos un asterisco antes del nombre de nuestra variable en el ejemplo siguiente veremos como asignamos el valor de memoria de por ejemplo num1 a nuestro puntero que es ejemplo:

EJEMPLO:

```
#include <stdio.h>
int main(){
{
    int num = 10;
    int *ejemplo;
    ejemplo = &num; // Asignación de la dirección de memoria de 'num' al puntero 'ejemplo'
    printf("El valor de num es: %dn", num);
    printf("La dirección de memoria de num es: %pn", &num);
    printf("El valor al que apunta ptr es: %dn", *ejemplo);
}
}
```

PUNTEROS Y VECTORES

1. Punteros:

- Un puntero es una variable que almacena la dirección de memoria de otra variable o dato.
- Puede apuntar a cualquier tipo de objeto, como estructuras o funciones.
- Se utilizan para manipular estructuras de datos, gestionar memoria dinámica y pasar argumentos por referencia en funciones.

2. Vectores:

- Un vector es un espacio continuo en memoria que almacena varios elementos del mismo tipo.
- El acceso a los elementos de un vector se realiza mediante corchetes.

DIFERENCIA ENTRE UN VECTOR Y UN PUNTERO

Definición:

Vector: Un vector es un conjunto de elementos del mismo tipo almacenados en posiciones de memoria consecutivas. Por ejemplo: int vec[5] crea un vector de 5 enteros.

Puntero: Un puntero es una variable que almacena la dirección de memoria de otro objeto. Por ejemplo, int *ptr declara un puntero a un entero.

Acceso a elementos:

Vector: Accedemos a los elementos del vector mediante índices (por ejemplo, vec[0], vec[1], etc.).

Puntero: Utilizamos la aritmética de punteros para acceder a los elementos apuntados por el puntero (por ejemplo, *ptr, *(ptr + 1), etc.).

Inicialización:

Vector: Se inicializa con valores directamente (por ejemplo, int vec[5] = {10, 20, 30, 40, 50}).

Puntero: Se inicializa asignándole la dirección de memoria de otro objeto (por ejemplo, int *pe; pe = vec;).

```
#include <stdio.h>
int main(){
{
    int a = 42;
    int *puntero = &a; // Asignamos la dirección de memoria de 'a' al printf("El valor de a es: %d\n", a);
    printf("El valor apuntado por *puntero es: %d\n", *puntero);
    printf("La dirección de memoria de *puntero es: %p\n", puntero);

    int miVector[6]; // Declaración de un vector de 6 posiciones
    miVector[0] = 10; // Asignamos un valor al primer elemento
}
}
```

ARITMÉTICA DE PUNTEROS

La aritmética de punteros te permite realizar operaciones matemáticas con punteros por ejemplo:

Si tienes un puntero p y le sumas o restas un valor X , estarás moviéndote X posiciones en la memoria desde la dirección original a la que apuntaba p .

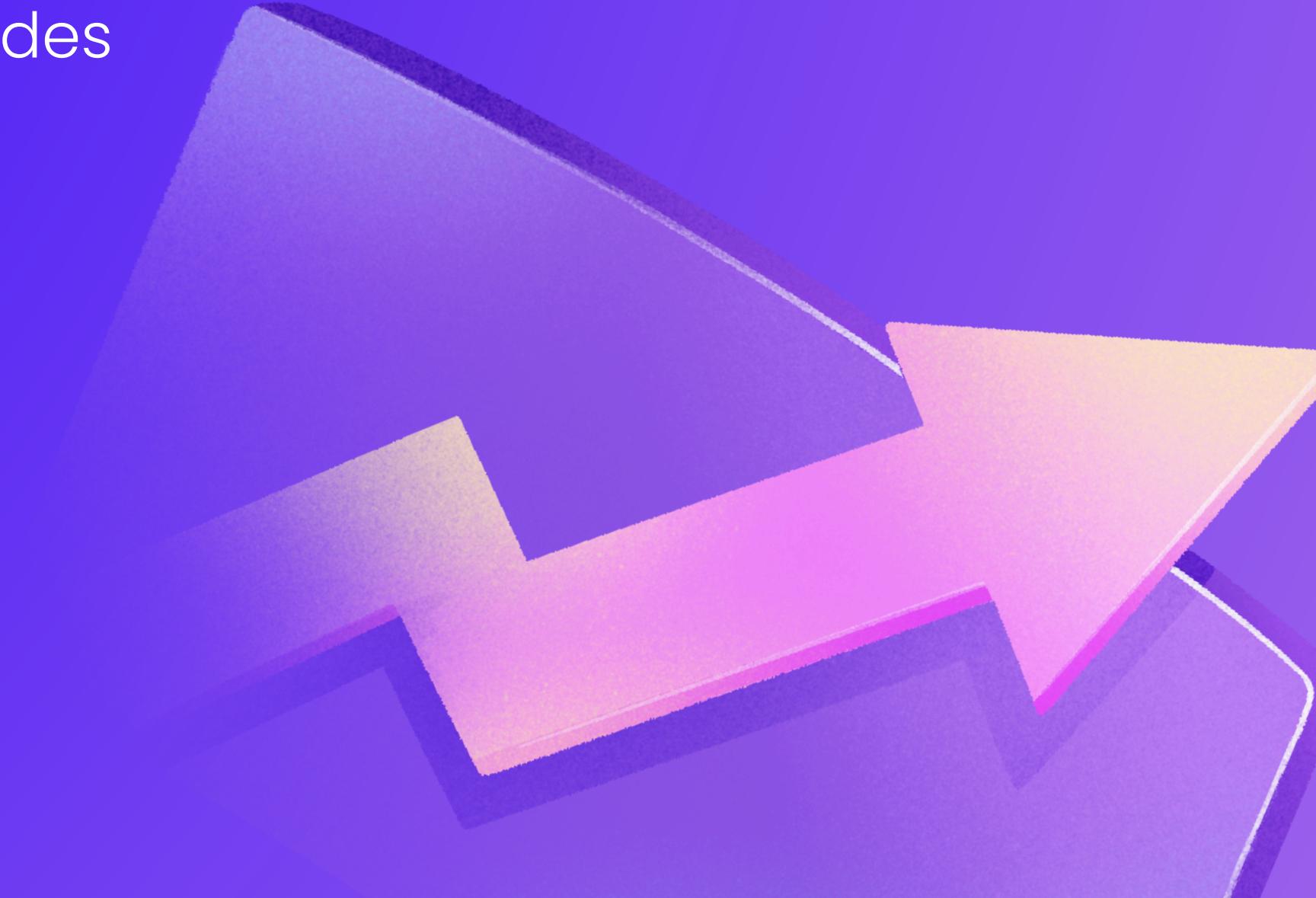


Por ejemplo, si tienes un arreglo `int vector[4]` y un puntero `int *p` que apunta a vector, al sumar 1 a `p`, estarás apuntando al siguiente elemento del arreglo.

Punteros a diferentes tipos de datos:

No todos los punteros son iguales. Por ejemplo, un puntero a `int` se mueve en trozos de 4 bytes mientras que un puntero a `char` se mueve solo un byte a la vez.

Imagina que elegir entre un carro y una bicicleta para recorrer la ciudad: ambos te llevan al mismo lugar, pero a diferentes velocidades



EJEMPLO

```
#include <stdio.h>

int main()
{
    int N = 4;

    int *ptr1, *ptr2;

    ptr1 = &N;
    ptr2 = &N;

    printf("Pointer ptr2 before Addition: ");
    printf("%p \n", ptr2);

    ptr2 = ptr2 + 3;
    printf("Pointer ptr2 after Addition: ");
    printf("%p \n", ptr2);

    return 0;
}
```

PUNTEROS A PUNTEROS

Puntero a puntero Un puntero es como un biovector es decir que almacena la direccion de un objeto, que puede ser a su vez otro puntero. La notacion de puntero a puntero requiere de un doble asterisco (**).

Los punteros también pueden representar el concepto de indirección múltiple. La regla de derivación por la que de un tipo T se deriva el tipo “puntero a T” también se puede aplicar a este nuevo puntero para obtener el tipo “puntero a puntero a T” definido como “T **”.

Estos punteros múltiples tienen el mismo tamaño que un puntero simple, la diferencia sólo está en el número de indirecciones para obtener el dato. El siguiente programa es un ejemplo en el que se manipulan punteros con varios niveles de indirección:

```
#include <stdio.h>
int main()
{
    int i;
    int *ptrToi;          //Puntero a entero
    int **ptrToPtrToi;    // Puntero a puntero a entero

    ptrToPtrToi = &ptrToi; // Puntero contiene dirección de puntero
    ptrToi = &i;          // Puntero contiene dirección de entero

    i = 10;               // Asignación directa
    *ptrToi = 20;          // Asignación indirecta
    **ptrToPtrToi = 30;    // Asignación con doble indirección

    return 0;
}
```

PUNTEROS A ESTRUCTURAS

Los punteros son variables que almacenan la dirección en memoria de otra variable. Se pueden especificar variables puntero que apunten a una estructura, lo que se utiliza frecuentemente cuando se quiere hacer una reserva dinámica de memoria para una estructura en un determinado momento de la ejecución del programa o cuando es necesario pasar la estructura por referencia a una función.

los punteros a estructuras nos permiten manipular y acceder a los campos de una estructura de manera eficiente.

```
#include <stdio.h>
int main()
{
    struct Estudiante {char nombre[20];
                      int edad;
    };
}
```

EJEMPLO

```
struct fechasEspeciales {  
    char tipo[30];  
    int dia;  
    int mes;  
    char horaDeInicio[8];  
    struct fechasEspeciales* siguiente;  
};
```

PUNTEROS A FUNCIONES

Las funciones con punteros que se utilizan ampliamente en C para tres propósitos principales:

1. Asignar nuevos objetos en el montón.
2. Pasar funciones a otras funciones.
3. Iterar sobre elementos en matrices u otras estructuras de datos.

Un puntero a función es una variable que almacena la dirección de una función, permitiendo llamar a la función a través del puntero²³.

la estructura a punteros
quiere decir a la forma en
la que nosotros
conectamos o usamos los
punteros con funciones
definicion rapida

gracias

```
#include <stdio.h>
int increment(int i)
{
    printf("increment %d by 1\n", i);
    return i + 1;
}
int decrement(int i)
{
    printf("decrement %d by 1\n", i);
    return i - 1;
}
int main(void)
{
    int num = 0;
    int (*fp)(int);
    fp = &increment;
    num = (*fp)(num);
    num = (*fp)(num);
    fp = &decrement;
    num = (*fp)(num);
    printf("num is now: %d\n", num);
    return 0;
}
```