Principios de Programación

Variables y Tipos de Datos

Entender qué representan en memoria `int`, `float` y `char`, cuándo conviene cada tipo y cómo afectan las operaciones.

Recordando: ¿Qué se trabajó la semana pasada?

Base previa

- Ya escribimos programas con `main`
- Ya usamos `printf()` y `scanf()`
- Ya aparecieron variables como `int numero1` y `float promedio`
- Ya vimos que el compilador exige cierta forma

Problemas a aclarar

- Por qué un dato usa `int` y otro usa `float`
- Qué significa realmente “declarar” una variable
- Qué pasa si el tipo elegido no coincide con el dato

Definición

Una variable es un nombre asociado a una región de memoria cuyo contenido puede cambiar durante la ejecución del programa.

EJEMPLO

```c
int edad;
edad = 20;
edad = 21;
printf("%d", edad);
```

Qué muestra

- `edad` tiene nombre y tipo
- El valor almacenado puede cambiar
- El programa vuelve a usar el mismo nombre para leerlo o mostrarlo

Declarar

Informar al compilador que una variable va a existir, qué nombre tendrá y qué tipo de dato representará.

SINTAXIS

```
tipo nombre_variable;
```

```
int edad;
float altura;
char inicial;
int numero1, numero2, numero3;
```

- **La variable debe declararse antes de usarse**
- El tipo y el nombre se indican en la declaración
- Se pueden declarar varias del mismo tipo en una línea

Idea clave

Una variable en C combina tres cosas: **un nombre**, **un tipo** y **una región de memoria** en la que el programa guarda un valor.

- `edad` es el **identificador**
- `int` indica que el valor será entero
- El programa reserva espacio para guardar ese entero

EJEMPLO MÍNIMO

```c
int edad;
```

Terminología

En el estándar de C, esa región de memoria donde "vive" el valor se describe como un **object**.

-> *No confundir con un **objet** de otro lenguaje*

Definición

Un identificador es el nombre que usa el programa para referirse a una variable, una constante, una función u otro elemento definido en el código.

EJEMPLOS

```
int edad;
float promedio;
char inicial;
```

- `edad` nombra una variable
- `promedio` nombra otra variable
- `inicial` nombra otra variable

Importancia

El identificador no guarda el valor por sí solo. Su función es permitir que el código pueda referirse a ese dato o a esa función con un nombre estable.

Reglas para Identificadores

Reglas

- Puede contener letras, dígitos y _
- Debe empezar con letra o _
- No puede tener espacios
- No puede coincidir con una palabra reservada

VÁLIDOS

```
int edad;
int
edad_estudiante;
int edadEstudiante;
int _contador;
int numero1;
```

INVÁLIDOS

```
int 2edad;
int edad
estudiante;
int int;
```

La regla no depende del significado del nombre. Aunque `2edad` describa bien el dato, sigue siendo inválido porque empieza con un dígito.

Definición

Las palabras reservadas son aquellas que ya tienen una función propia dentro del lenguaje. Por eso no se pueden reutilizar como identificadores.

EJEMPLOS FRECUENTES

```
int, float, char, void
if, else, while, for, do
return, break, continue
#define
```

Error: usar una palabra reservada como nombre

CÓDIGO CON ERROR

```
int int = 5;
float if = 3.5;
```

MENSAJE / PROBLEMA

`int` e `if` ya significan algo en C. El compilador no puede interpretarlos además como nombres elegidos por el programador.

CORRECCIÓN

```
int numero = 5;
float precio = 3.5;
```

Criterios

- Preferir nombres descriptivos
- Reservar `x`, `a` o `b` para ejemplos muy breves
- Mantener un estilo consistente
- En nombres compuestos, usar _ o camelCase

MEJOR OPCIÓN

```
int edad;
float promedio_calificaciones;
int numeroDeEstudiantes;
```

A EVITAR

```
int x;
int a1;
int dato2;
```

Consistencia

`edad_estudiante` y `edadEstudiante` pueden funcionar. Lo importante es no mezclar estilos sin criterio dentro del mismo programa.

¿Por qué Necesitamos Tipos de Datos?

Cada tipo de dato define **qué clase de valores entran**, **cuánto espacio usan** y **qué operaciones tienen sentido** sobre ellos.

Representación

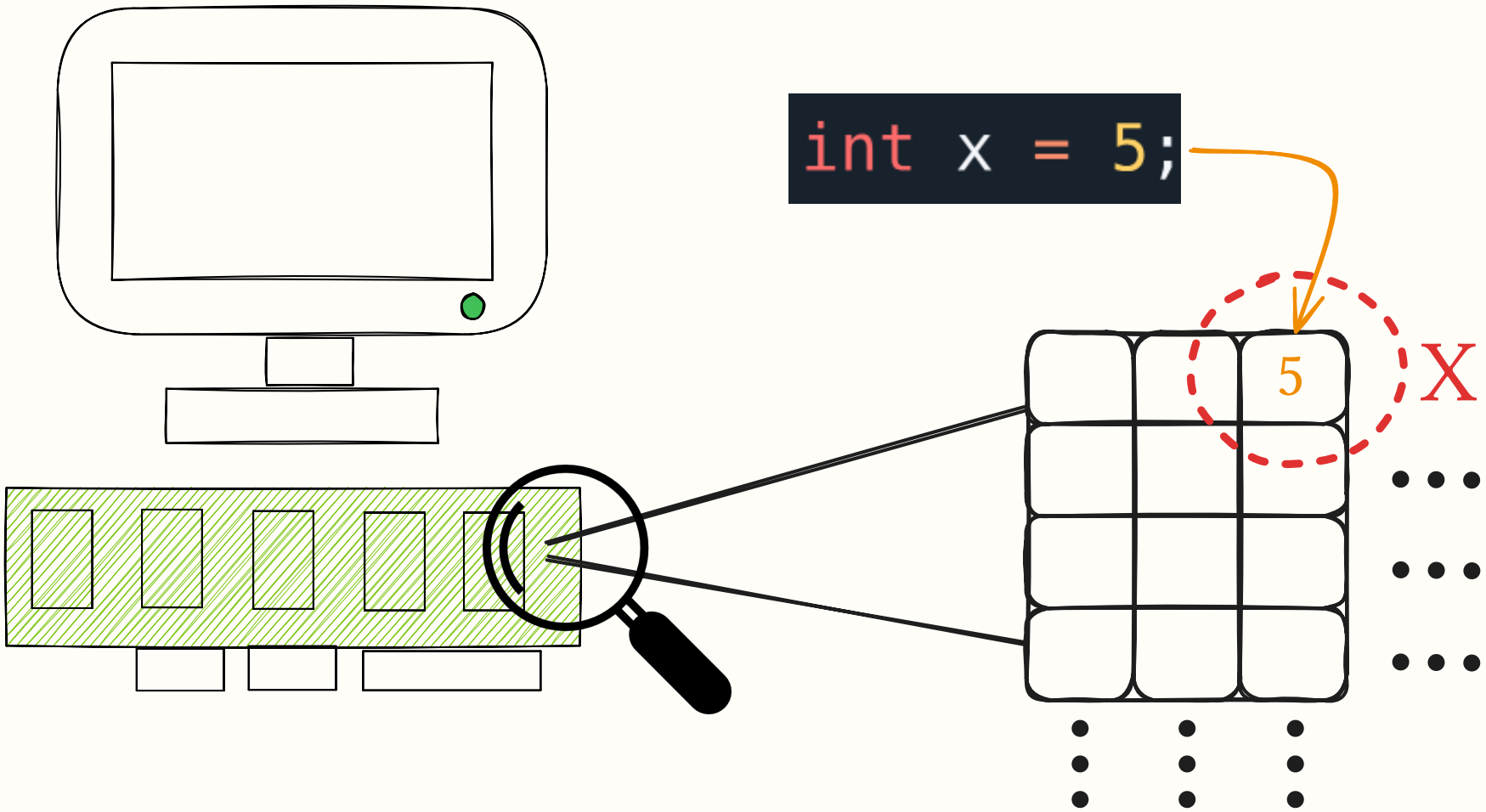
No es lo mismo guardar `20` que `3.14` o `'A'`.

Memoria

El tipo influye en cuánto espacio ocupa el dato.

Operaciones

El tipo también condiciona el resultado de expresiones y conversiones.



Lectura de la imagen

`x` es el nombre, `int` define el tipo y el valor `5` queda guardado en una ubicación de memoria durante la ejecución.

ALCANCE EN BLOQUES

```c
int main() {
    int x = 5;      // x vive en todo main
    {
        int y = 10; // y vive solo aquí
    }
    // y ya no existe
    return 0;
}
```

1 · Bloque externo

`x` existe mientras se ejecuta el cuerpo de `main`.

2 · Bloque interno

`y` aparece al entrar al bloque interno y deja de existir al salir.

3 · Regla mental

Además del nombre y el tipo, toda variable tiene un **lugar donde existe** y un **tiempo de vida**.

Ejercicio: ¿qué variables existen?

FRAGMENTO

```
int main() {
    int a = 1;
    {
        int b = 2;
        a = a + b;
    }
    // Punto A
    return 0;
}
```

- Dentro del bloque interno, ¿qué variables existen?
- En // **Punto A**, ¿cuáles siguen disponibles?

- En el bloque interno existen `a` y `b`
- En // **Punto A** solo existe `a`
- `b` desaparece cuando termina su bloque

Definición

Una constante es un valor que el programa trata como fijo. Puede aparecer escrito directamente o puede recibir un nombre para reutilizarse con claridad.

CON `#define`

```
#define PI 3.14159
#define MAX_ESTUDIANTES 100
```

CON `const`

```
const int MAX_INTENTOS = 3;
const float DESCUENTO = 0.10;
```

Ventajas

- Hacen el código más legible
- Evitan repetir el mismo valor “mágico”
- Facilitan cambiar el valor en un solo lugar

Tipos básicos

- `int`: enteros
- `float`: decimales
- `char`: un carácter
- `double`: decimal con mayor precisión

Usos típicos

- `int`: edades, cantidades, contadores
- `float`: precios, medidas, promedios
- `char`: iniciales, respuestas cortas
- `double`: cálculos que necesitan más precisión

Criterio

Elegir el tipo no es un detalle de sintaxis. Determina qué valores entran y cómo se comportan las operaciones.

Tabla de Tipos Básicos

Tipo	Tamaño típico	Ejemplo	Uso frecuente
`int`	4 bytes	`5`, `-10`, `1000`	Cantidades enteras
`float`	4 bytes	`3.14`, `-2.5`	Medidas y promedios
`char`	1 byte	`'A'`, `'5'`, `'?'`	Un carácter
`double`	8 bytes	`3.14159`	Mayor precisión decimal

EJEMPLO

```
int edad = 20;
float altura = 1.75;
char inicial = 'M';
double pi = 3.14159265359;
```

Ejemplo: Tipos Correctos e Incorrectos

TIPOS ADECUADOS

```
int edad = 20;
float altura = 1.75;
char inicial = 'M';
```

TIPOS PROBLEMÁTICOS

```
int edad = 1.75;
float altura = 'M';
```

Qué pasa realmente

- `int edad = 1.75;` guarda `1`
- Se pierde la parte decimal porque `int` no representa fracciones

Otro error semántico

- `float altura = 'M';` termina usando el código ASCII
- El valor numérico puede existir, pero no representa una altura coherente

Tipo int - Números Enteros

Características

- Representa números enteros
- Puede ser positivo o negativo
- Suele ocupar 4 bytes
- No guarda parte decimal

EJEMPLOS

```
int edad = 20;
int cantidad = 100;
int temperatura = -5;
int resultado = 0;
int numero_estudiantes = 25;
```

OPERACIONES CON `int`

```
int a = 10;
int b = 3;
int suma = a + b;       // 13
int resta = a - b;      // 7
int producto = a * b;   // 30
int cociente = a / b;   // 3
int resto = a % b;      // 1
```

Lectura

- `+`, `-`, `*`, `/` y `%` operan con enteros
- La división entre enteros conserva solo el cociente entero
- `%` devuelve el resto

División Entera

EJEMPLO

```
int a = 10;
int b = 3;
int resultado = a / b;  // 3
```

1 · Ambos son `int`

La operación se resuelve como división entera.

2 · Se descarta la fracción

El resultado almacenado es `3`, no `3.33`.

3 · Para decimal hace falta otro tipo

Más adelante se verá cómo interviene `float`.

Operador Módulo (%)

RESTO DE UNA DIVISIÓN

```
int a = 10;
int b = 3;
int resto = a % b;  // 1
```

Qué significa

- `10 / 3` resulta `3`
- `%` devuelve el sobrante de esa cuenta
- En este caso, el resto es `1`

EJEMPLO PRÁCTICO

```
int horas = 125;
int dias = horas / 24;
int horas_restantes = horas % 24;
```

Uso inmediato

Cuantos días son 125 horas? y cuantas horas restantes?

Tipo float - Números Decimales

Características

- Representa números con parte decimal
- Puede ser positivo o negativo
- Suele ocupar 4 bytes
- Tiene precisión limitada

Cuándo usarlo

Precios, medidas, promedios y cualquier resultado que necesite conservar fracciones.

EJEMPLOS

```
float precio = 99.99;
float altura = 1.75;
float temperatura = -5.5;
float promedio = 85.5;
float peso = 68.5;
```

NOTACIÓN CIENTÍFICA

```
float grande = 1.5e6;
float pequeno = 2.5e-3;
```

OPERACIONES CON `float`

```
float a = 10.5;
float b = 3.2;
float suma = a + b;
float resta = a - b;
float producto = a * b;
float cociente = a / b;
```

Punto delicado

La división solo conserva parte decimal si **al menos uno** de los operandos participa como decimal.

COMPARACIÓN

```
float r1 = 10 / 3;    // 3.0
float r2 = 10.0 / 3;  // 3.333...
float r3 = 10 / 3.0;  // 3.333...
```

Características

- Guarda un solo carácter
- Puede ser letra, dígito, símbolo o espacio
- Suele ocupar 1 byte
- Se escribe entre **comillas simples**

Usos

Iniciales, respuestas breves y cualquier caso donde el programa necesite almacenar un único carácter.

EJEMPLOS

```
char letra = 'A';
char digito = '5';
char simbolo = '?';
char espacio = ' ';
char respuesta = 'S';
```

Diferencia importante

Un `char` usa comillas simples. Las comillas dobles se reservan para texto.

OPERACIONES CON char

```c
char letra1 = 'A';
char letra2 = 'B';

int codigo = letra1;       // 65
char siguiente = letra1 + 1; // 'B'
```

Por qué funciona

- Un `char` se almacena con un valor numérico
- En ASCII, `'A' = 65`, `'B' = 66`, `'C' = 67`
- Por eso podemos sumar, comparar o imprimir su código

EJEMPLO DE SALIDA

```c
char inicial = 'M';
printf("Mi inicial es: %c\n", inicial);
printf("Su código ASCII es: %d\n", inicial);
```

SALIDA

```
Mi inicial es: M
Su código ASCII es: 77
```

Idea a retener

No hace falta memorizar toda la tabla ASCII. Lo importante es entender que un `char` tiene representación numérica y que eso explica estas conversiones.

0-31

Dec	Char
0	NUL
1	SOH
2	STX
3	ETX
4	EOT
5	ENQ
6	ACK
7	BEL
8	BS
9	TAB
10	LF
11	VT
12	FF
13	CR
14	SO
15	SI
16	DLE
17	DC1
18	DC2
19	DC3
20	DC4
21	NAK
22	SYN
23	ETB
24	CAN
25	EM
26	SUB
27	ESC
28	FS
29	GS
30	RS
31	US

32-63

Dec	Char
32	SP
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?

64-95

Dec	Char
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_

96-127

Dec	Char
96	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	|
125	}
126	~
127	DEL

Controles y especiales · Dígitos 0-9 · Mayúsculas A-Z · Minúsculas a-z

Ejemplo: Declaración de Variables

DECLARAR, ASIGNAR Y USAR

```c
#include <stdio.h>

int main() {
    int edad;
    float altura;
    char inicial;

    edad = 20;
    altura = 1.75;
    inicial = 'M';

    printf("Edad: %d\n", edad);
    printf("Altura: %.2f\n", altura);
    printf("Inicial: %c\n", inicial);

    return 0;
}
```

Qué muestra

Primero se declaran las variables y recién después se les asignan valores y se usan en `printf`.

ERROR FRECUENTE

```c
int main() {
    edad = 20;
    int edad;
}
```

Error: variable usada antes de declararse

CÓDIGO CON ERROR

```
int main() {
    edad = 20;
    int edad;
}
```

MENSAJE / PROBLEMA

El compilador necesita conocer **el tipo** y **la existencia** de edad antes de permitir cualquier uso de ese nombre.

CORRECCIÓN

```
int main() {
    int edad;
    edad = 20;
}
```

DOS PASOS

```
int edad;
edad = 20;
```

AL DECLARAR

```
int edad = 20;
float altura = 1.75;
char inicial = 'M';
```

Idea práctica

Inicializar significa darle un valor inicial a la variable. Puede hacerse después de declarar, pero cuando ya sabemos el valor conviene hacerlo en la misma línea.

MÚLTIPLES VARIABLES

```
int a = 5, b = 10, c = 15;
int x = 1, y, z = 3;
```

Error: usar una variable sin valor inicial

CÓDIGO CON ERROR

```c
int suma;
printf("%d\n", suma);
```

MENSAJE / PROBLEMA

`suma` existe, pero todavía no tiene un valor confiable. Leerla en ese estado produce un resultado impredecible.

CORRECCIÓN

```c
int suma = 0;
printf("%d\n", suma);
```

Programa: Información de Estudiante

Un ejemplo mínimo que combina `int`, `float` y `char` en un mismo programa y obliga a usar el especificador correcto para cada caso.

CÓDIGO COMPLETO

```c
#include <stdio.h>

int main() {
    int edad = 20;
    float promedio = 85.5;
    char inicial = 'J';

    printf("Edad: %d años\n", edad);
    printf("Promedio: %.2f\n", promedio);
    printf("Inicial: %c\n", inicial);

    return 0;
}
```

Lectura paso a paso

1. `edad` se declara como `int` y recibe `20`
2. `promedio` se declara como `float` y recibe `85.5`
3. `inicial` se declara como `char` y recibe `'J'`
4. Cada `printf` usa el especificador compatible con el tipo

SALIDA

```
Edad: 20 años
Promedio: 85.50
Inicial: J
```

Detalle importante

`%.2f` muestra el `float` con dos decimales. La idea no es memorizar símbolos aislados, sino asociar cada especificador con el tipo correcto.

Qué aporta una traza

Obliga a seguir el programa paso a paso y deja visible cómo cambian las variables y qué salida aparece en cada momento.

TRAZA · EDAD 20, PROMEDIO 85.5, INICIAL "J"

Paso	edad	promedio	inicial	Salida
Inicio	—	—	—	—
Inicialización	20	85.5	"J"	—
`printf edad`	20	85.5	"J"	Edad: 20 años
`printf promedio`	20	85.5	"J"	Promedio: 85.50
`printf inicial`	20	85.5	"J"	Inicial: J

REFERENCIA

Operador	Uso	Ejemplo	Resultado
+	Suma	5 + 3	8
-	Resta	5 - 3	2
*	Producto	5 * 3	15
/	División	5 / 3	Entero o decimal según el tipo
%	Resto	5 % 3	2

EJEMPLO

```
int a = 10, b = 3;
int suma = a + b;
int resta = a - b;
int producto = a * b;
int cociente = a / b;
int resto = a % b;
```

Operadores de Asignación Compuesta

FORMA DESARROLLADA

```
x = x + 5;
x = x - 3;
x = x * 2;
x = x / 4;
```

FORMA ABREVIADA

```
x += 5;
x -= 3;
x *= 2;
x /= 4;
```

Idea práctica

La variable aparece una sola vez a la izquierda, pero el significado sigue siendo “operar con el valor actual y guardar el resultado en la misma variable”.

Incremento y Decremento

EJEMPLO

```
int x = 5;
x++;
x--;
```

Equivalencias

- `x++` equivale a sumar `1`
- `x--` equivale a restar `1`
- También puede escribirse con `+= 1` o `-= 1`

Precedencia de Operadores

Orden básico

1. Paréntesis ( )
2. Multiplicación, división y módulo
3. Suma y resta

EJEMPLO CLAVE

```
int resultado1 = 2 + 3 * 4;   // 14
int resultado2 = (2 + 3) * 4; // 20
```

Precedencia - Explicación

1 · Multiplicar

En `2 + 3 * 4`, primero se calcula `3 * 4 = 12`.

2 · Sumar

Luego la expresión pasa a ser `2 + 12 = 14`.

3 · Reordenar

Si aparecen paréntesis, el orden cambia y el resultado puede pasar a `20`.

VERSIÓN CON PARÉNTESIS

```
int resultado2 = (2 + 3) * 4;   // 20
```

EJEMPLOS RESUELTOS

Expresión	Lectura
`10 + 5 * 2`	Primero `5 * 2 = 10`, luego `10 + 10 = 20`
`(10 + 5) * 2`	Primero `10 + 5 = 15`, luego `15 * 2 = 30`
`10 / 2 + 3`	Primero `10 / 2 = 5`, luego `5 + 3 = 8`
`10 / (2 + 3)`	Primero `2 + 3 = 5`, luego `10 / 5 = 2`
`10 % 3 + 2`	Primero `10 % 3 = 1`, luego `1 + 2 = 3`

Recomendación: Usar Paréntesis

MENOS CLARO

```
int x = a + b * c - d;
```

MÁS CLARO

```
int x = a + (b * c) - d;
```

Recomendación

Si una expresión puede generar duda al leerla, conviene agregar paréntesis. La intención queda explícita y el código se vuelve más fácil de revisar.

División Entera vs Decimal

ENTRE ENTEROS

```
int a = 10;
int b = 3;
int resultado = a / b;   // 3
```

CON DECIMALES

```
float a = 10.0;
int b = 3;
float resultado = a / b; // 3.333...
```

Si ambos operandos son `int`, la división es entera y descarta la parte decimal. Para obtener un resultado decimal, al menos uno debe participar como `float`.

Problema

Leer dos números y calcular suma, resta, producto y división, cuidando que la división no pierda la parte decimal.

Ejemplo 1: Calculadora Básica

CALCULADORA BÁSICA

```c
#include <stdio.h>

int main() {
    int numero1, numero2;
    int suma, resta, producto, division;

    printf("Ingrese primer número: ");
    scanf("%d", &numero1);

    printf("Ingrese segundo número: ");
    scanf("%d", &numero2);

    suma = numero1 + numero2;
    resta = numero1 - numero2;
    producto = numero1 * numero2;
    division = numero1 / numero2;

    printf("Suma: %d\n", suma);
    printf("Resta: %d\n", resta);
    printf("Producto: %d\n", producto);
    printf("División: %d\n", division);

    return 0;
}
```

Ejemplo 1: Análisis

Qué hace el programa

- Declara variables para entradas y resultados
- Lee dos enteros con `scanf`
- Calcula las cuatro operaciones
- Usa `int` por lo tanto la división NO conserva decimales

EJECUCIÓN POSIBLE

```
Ingrese primer número: 10
Ingrese segundo número: 3
Suma: 13
Resta: 7
Producto: 30
División: 3
```

Problema

Calcular el área y el perímetro de un rectángulo usando medidas que pueden ser decimales.

CÓDIGO

```c
#include <stdio.h>

int main() {
    float base, altura;
    float area, perimetro;

    printf("Ingrese la base: ");
    scanf("%f", &base);

    printf("Ingrese la altura: ");
    scanf("%f", &altura);

    area = base * altura;
    perimetro = 2 * (base + altura);

    printf("Área: %.2f\n", area);
    printf("Perímetro: %.2f\n", perimetro);
    return 0;
}
```

Ejemplo 2: Análisis

Lectura

- `float` permite ingresar medidas con decimales
- `area = base * altura`
- `perimetro = 2 * (base + altura)`
- `%.2f` limita la salida a dos decimales

EJECUCIÓN POSIBLE

```
Ingrese la base: 5.5
Ingrese la altura: 3.2
Área: 17.60
Perímetro: 17.40
```

Problema

Convertir una cantidad total de segundos en horas completas, minutos y segundos restantes.

CÓDIGO

```c
#include <stdio.h>

int main() {
    int segundos_totales;
    int horas, minutos, segundos;

    printf("Ingrese segundos: ");
    scanf("%d", &segundos_totales);

    horas = segundos_totales / 3600;
    minutos = (segundos_totales % 3600) / 60;
    segundos = segundos_totales % 60;

    printf("%d segundos = ", segundos_totales);
    printf("%d horas, %d minutos, %d segundos\n", horas, minutos, segundos);
    return 0;
}
```

Ejemplo 3: Análisis

1 · Horas

`segundos_totales / 3600` calcula cuántas horas completas entran.

2 · Minutos

Primero se toma el resto frente a `3600` y luego se divide entre `60`.

3 · Segundos

Un último `% 60` deja solo los segundos que sobran.

EJECUCIÓN POSIBLE

```
Ingrese segundos: 7325
7325 segundos = 2 horas, 2 minutos, 5
segundos
```

Error 1 · Variable sin declarar

CÓDIGO CON ERROR

```
int main() {
    x = 5;
    return 0;
}
```

MENSAJE / PROBLEMA

El compilador no puede usar `x` si todavía no recibió su declaración y su tipo.

CORRECCIÓN

```
int x;
```

Error 2 · Variable sin inicializar

CÓDIGO CON ERROR

```c
int suma;
printf("%d\n", suma);
```

MENSAJE / PROBLEMA

La variable existe, pero todavía no tiene un valor confiable para leerse.

CORRECCIÓN

```c
int suma = 0;
```

Error 3 · Nombre inválido

CÓDIGO CON ERROR

```
int 2edad;
int edad estudiante;
```

MENSAJE / PROBLEMA

Un identificador no puede empezar con un número ni contener espacios.

CORRECCIÓN

```
int edad2;
int edad_estudiante;
```

Error 4 · División resuelta como entera

CÓDIGO CON ERROR

```
int a = 10;
int b = 3;
float resultado = a / b;
```

MENSAJE / PROBLEMA

La división `a / b` se calcula primero entre enteros. Recién después ese `3` pasa a `3.0`, por lo que la fracción ya se perdió.

CORRECCIÓN

```
float resultado = (float)a / b;
```

Error 5 · Especificador incorrecto

CÓDIGO CON ERROR

```c
int edad = 20;
printf("%f\n", edad);
```

MENSAJE / PROBLEMA

`%f` corresponde a valores decimales. Si la variable es `int`, el especificador compatible es `%d`.

CORRECCIÓN

```c
printf("%d\n", edad);
```

Error 6 · Palabra reservada

CÓDIGO CON ERROR

```
int int = 5;
```

MENSAJE / PROBLEMA

`int` ya es una palabra del lenguaje y no puede reutilizarse como identificador.

CORRECCIÓN

```
int numero = 5;
```

Resumen: Identificadores y Variables

Identificadores

- Nombran variables, constantes y funciones
- Deben respetar reglas de escritura
- No pueden usar palabras reservadas

Variables y constantes

- Las variables cambian durante la ejecución
- Las constantes representan valores fijos
- Declarar antes de usar evita errores

Tipos

- `int` para enteros
- `float` para decimales
- `char` para un carácter
- El tipo correcto depende del dato y de la operación

Resumen: Operadores y Buenas Prácticas

Operadores

- `+`, `-`, `*`, `/`, `%`
- Precedencia básica
- División entera vs decimal

Buenas prácticas

- Nombres descriptivos
- Variables inicializadas
- Paréntesis para claridad
- Tipos coherentes con el dato

Próxima clase

- Selección en C
- `if`, `if-else` y `else if`
- Operadores de comparación y lógicos
- `switch-case`

Ejercicio para Practicar

Consigna

1. Leer tres calificaciones
2. Calcular el promedio
3. Mostrar el resultado con dos decimales

BASE

```c
#include <stdio.h>

int main() {
    float calif1, calif2, calif3;
    float promedio;

    // Escribir el código aquí
    return 0;
}
```

Ejercicio: Variantes y Recordatorios

EJECUCIÓN POSIBLE

```
Ingrese primera calificación: 85.5
Ingrese segunda calificación: 90.0
Ingrese tercera calificación: 78.5
El promedio es: 84.67
```

Variantes

- Promedio de 4 calificaciones
- Promedio ponderado
- Mostrar también la suma total

Recursos y Próxima Clase

Bibliografía

- **El lenguaje de programación C** — Kernighan & Ritchie
- **Cómo programar en C/C++** — Deitel & Deitel
- Capítulo 2: tipos, operadores y expresiones

Próxima clase

- Selección en C
- `if`, `if-else` y `else if`
- Operadores de comparación y lógicos
- `switch-case`