

# Programación 1

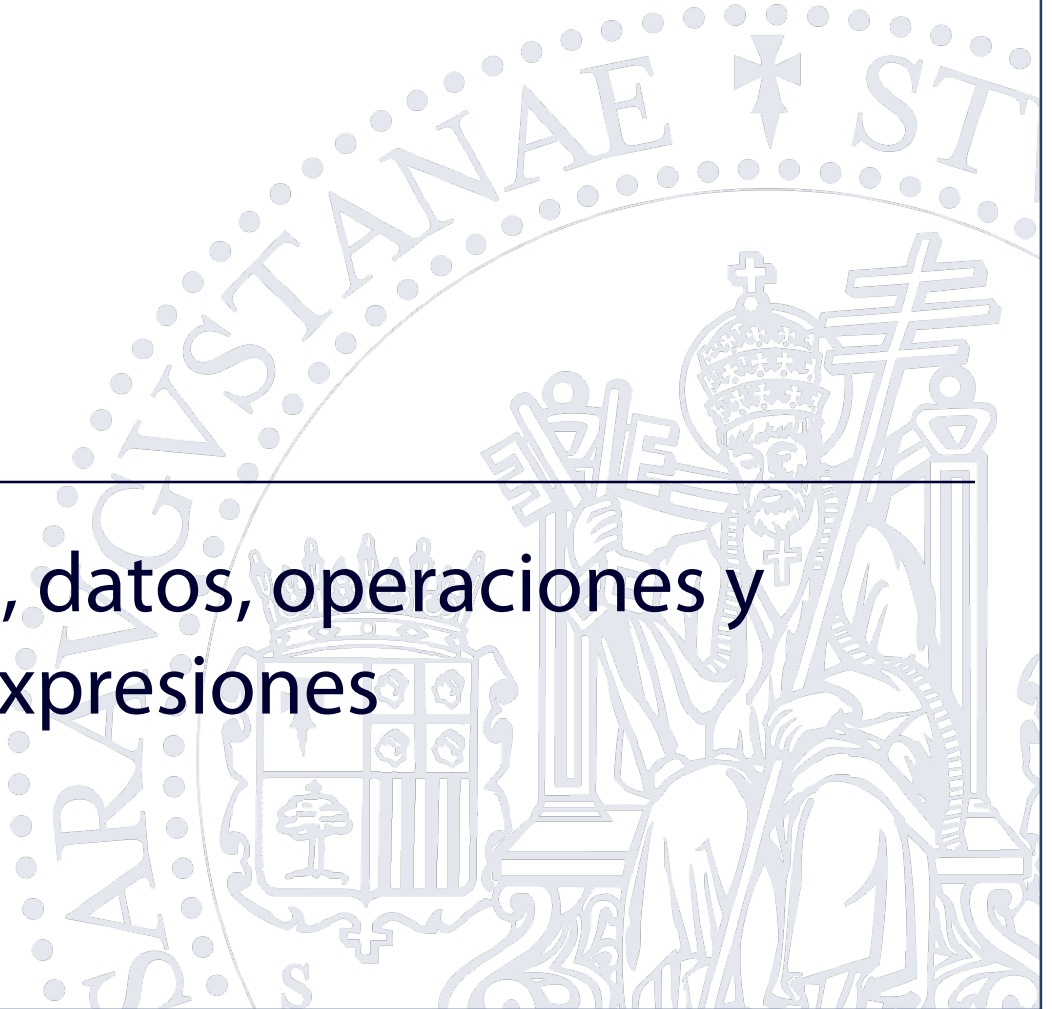
## Tema 3

---

### Información, datos, operaciones y expresiones



Escuela de  
Ingeniería y Arquitectura  
**Universidad Zaragoza**





# Índice

---

- ❑ Datos y tipos de datos
- ❑ Datos primitivos en C++
- ❑ Expresiones e instrucción de asignación

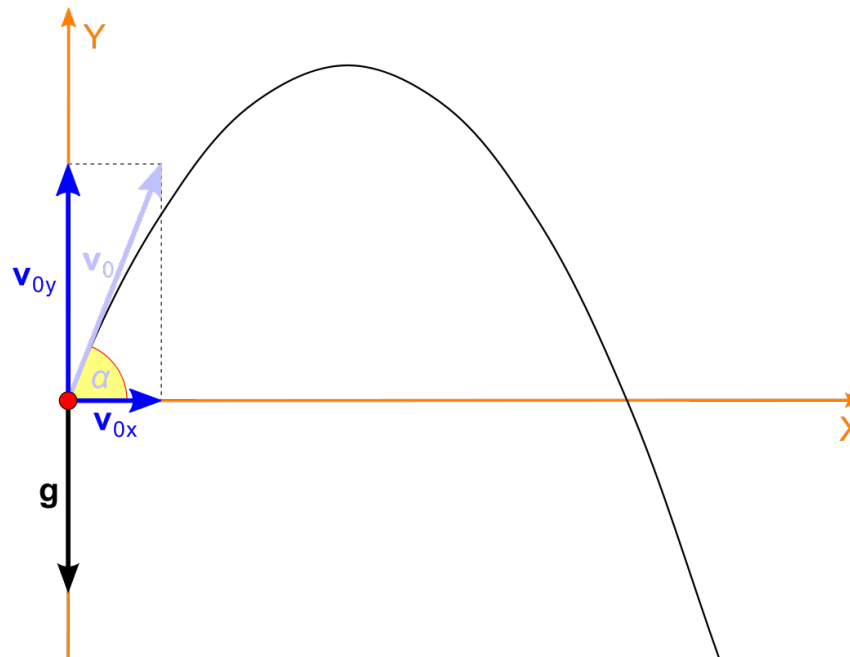
# Datos y tipos de datos

---

- ❑ Problema → información → abstracción → datos
- ❑ Cada dato tiene un **valor**
- ❑ Con los datos se realizan cálculos y operaciones

# Ejemplo

- ¿Qué información hace falta para resolver el problema de la trayectoria que sigue un proyectil?





# Ejemplo

## Lanzamiento de un proyectil

- Información relevante
  - Velocidad inicial  $v_0$
  - Ángulo de tiro  $\alpha$
  - Posición inicial  $\vec{r}$
  - Aceleración de la gravedad  $g$
  - Coeficiente de rozamiento  $\mu$
- Información no relevante  
(pero de tipos de datos distintos a  $\mathbb{R}$ )
  - Número de perdigones  $n$
  - ¿Es de día o de noche?
  - Nombre del fabricante de los perdigones
  - Datos personales de la persona que dispara

# Datos en C++

---

- Tipos primitivos de datos
  - No derivan de otros tipos de datos
  - Dominio finito de valores
  - Codificación binaria definida
  - Sintaxis para representar sus valores
  - Operaciones predefinidas
- Tipos estructurados

# Tipos primitivos en C++

---

- Enteros
  - short, **int**, long, long long
  - unsigned short, **unsigned int**, unsigned long, unsigned long long
- Reales
  - float, **double**, long double
- Booleanos
  - **bool**
- Caracteres
  - **char**

# Naturales

---

- Dominio de valores (GCC y MinGW)
  - Subconjunto de  $\mathbb{N}$ 
    - `unsigned int`  **$0..4 \times 10^9$**
- Representación externa en C++
  - 0 1 6 2541 45000163 ...
- Codificación (GCC y MinGW)
  - Binaria (32 bits)



# Enteros

---

- Dominio de valores (GCC y MinGW)
  - Subconjunto de  $\mathbb{Z}$ 
    - `int`  $-2 \times 10^9 \dots 2 \times 10^9$
- Representación externa en C++
  - `0 1 -1 6 2541 -12022 ...`
- Codificación (GCC y MinGW)
  - Complemento a dos (32 bits)

# Tipos reales

---

- Dominio de valores (GCC y MinGW)
  - Subconjunto de  $\mathbb{R}$ 
    - `double`  $-1.79769313 \times 10^{308} .. +1.79769313 \times 10^{308}$
- Representación externa en C++
  - `0.0   0.5   -1.75`  
`3.14159265358979323846`  
`6.022e23   -1.602e-19`
- Codificación (GCC y MinGW)
  - IEEE 754 (64 bits)

# ***Booleanos***

---

- **bool**
- Dominio de valores
  - {falso, cierto}
- Representación externa en C++
  - **false    true**
- Codificación
  - 8 bits (1 byte)

# Caracteres

- **char**
- **Dominio de valores**
  - 96 caracteres del alfabeto inglés
    - Letras
    - Dígitos
    - Signos de puntuación
    - Otros símbolos
  - 32 *caracteres* de control
  - 128 caracteres dependientes de la codificación

	0	@	P	`	p
!	1	A	Q	a	q
"	2	B	R	b	r
#	3	C	S	c	s
\$	4	D	T	d	t
%	5	E	U	e	u
&	6	F	V	f	v
'	7	G	W	g	w
(	8	H	X	h	x
)	9	I	Y	i	y
*	:	J	Z	j	z
+	;	K	[	k	{
,	<	L	\	l	
-	=	M	]	m	}
.	>	N	^	n	~
/	?	O	_	o	

# Caracteres

## □ Representación externa en C++

- 'a' 'A' 'b' 'B' 'z' 'Z'
- '0' '1' '2' '3' '4' '5' '6' '7'  
'8' '9'
- '+' '-' '\*' '/' '<' '=' '>'
- '(' ')' '[' ']' '{' '}'
- '#' '\$' '%' '&' ',' '.' ':' ';'  
'!' '?' '@' '^' '\_' '`' '|' '~'
- '"' '\'

# Operaciones (datos primitivos)

---

- Unarias (enteros y reales)
  - +, -
- Aritméticas (enteros y reales)
  - +, -, \*, /, %
- Lógicas (*booleanos*)
  - !, &&, ||
- Relacionales (enteros, reales, caracteres, *booleanos*, ...)
  - ==, !=
  - >, >=, <, <=

# Datos constantes y variables

---

- Constantes literales
  - 0, 25, -8, 3.14159, **true**, **false**, 'a', 'Z', "Universidad de Zaragoza"
- Constantes simbólicas
  - **const int** MAXIMO = 1000;
  - **const int** ANCHO = 9;
  - **const double** PI = 3.141592653589793;
- Variables
  - Variables locales
  - Parámetros de una función

# Variables

---

- ❑ Datos cuyo **valor** puede variar entre ejecuciones
  - O incluso en la misma ejecución
- ❑ Siempre tienen un valor asociado
- ❑ En C++ tienen asociado un tipo no modificable
- ❑ En C++ se implementan ocupando la cantidad de memoria del computador necesaria y codificando el valor de la variable de acuerdo con el esquema de codificación asociado al tipo de datos de la variable.





# Variables



00000101	@1800
10110110	@1801
01000110	@1802
10101110	@1803
10101000	@1804
00110001	@1805
01101011	@1806
00001011	@1807
01110001	@1808
10101100	@1809
10011011	@1810
10001111	@1811
01110001	@1812
11101110	@1813
11000110	@1814



# Variables

```
int a;
```

00000101	@1800
10110110	@1801
01000110	@1802
10101110	@1803
10101000	@1804
00110001	@1805
01101011	@1806
00001011	@1807
01110001	@1808
10101100	@1809
10011011	@1810
10001111	@1811
01110001	@1812
11101110	@1813
11000110	@1814



# Variables

```
int a;
```

a	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
	10101000	@1804
	00110001	@1805
	01101011	@1806
	00001011	@1807
	01110001	@1808
	10101100	@1809
	10011011	@1810
	10001111	@1811
	01110001	@1812
	11101110	@1813
	11000110	@1814



# Variables

```
int a;  
int b1 = 3;
```

a

00000101	@1800
10110110	@1801
01000110	@1802
10101110	@1803
10101000	@1804
00110001	@1805
01101011	@1806
00001011	@1807
01110001	@1808
10101100	@1809
10011011	@1810
10001111	@1811
01110001	@1812
11101110	@1813
11000110	@1814

# Variables

```
int a;  
int b1 = 3;
```

a	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
b1	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
	01110001	@1808
	10101100	@1809
	10011011	@1810
	10001111	@1811
	01110001	@1812
	11101110	@1813
	11000110	@1814

# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
	01110001	@1808
	10101100	@1809
	10011011	@1810
	10001111	@1811
	01110001	@1812
	11101110	@1813
	11000110	@1814

# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
	01110001	@1812
	11101110	@1813
	11000110	@1814

# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;  
char c1;
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
	01110001	@1812
	11101110	@1813
	11000110	@1814



# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;  
char c1;
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
<b>c1</b>	01110001	@1812
	11101110	@1813
	11000110	@1814

# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;  
char c1;  
char c2 = 'A';
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
<b>c1</b>	01110001	@1812
	11101110	@1813
	11000110	@1814

# Variables

```
int a;
int b1 = 3;
int b2 = -3;
char c1;
char c2 = 'A';
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
<b>c1</b>	01110001	@1812
<b>c2</b>	01000001	@1813
	11000110	@1814

# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;  
char c1;  
char c2 = 'A';  
bool d = (b1 == 8);
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
<b>c1</b>	01110001	@1812
<b>c2</b>	01000001	@1813
	11000110	@1814

# Variables

```
int a;
int b1 = 3;
int b2 = -3;
char c1;
char c2 = 'A';
bool d = (b1 == 8);
```

<b>a</b>	00000101	@1800
	10110110	@1801
	01000110	@1802
	10101110	@1803
<b>b1</b>	00000000	@1804
	00000000	@1805
	00000000	@1806
	00000011	@1807
<b>b2</b>	11111111	@1808
	11111111	@1809
	11111111	@1810
	11111101	@1811
<b>c1</b>	01110001	@1812
<b>c2</b>	01000001	@1813
<b>d</b>	00000000	@1814

# Variables

```
int a;  
int b1 = 3;  
int b2 = -3;  
char c1;  
char c2 = 'A';  
bool d = (b1 == 8);
```

a	?
b1	3
b2	?
c1	?
c2	'A'
d	false

# Declaración de variables

---

- Datos de tipos primitivos
  - `int i, j, k;`
  - `unsigned m, n;`
  - `char c1, c2;`
  - `bool b;`
  - `double r1, r2, r3;`

# Declaración de variables

---

## □ Datos de tipos primitivos

- `int i = 3,  
j = 0,  
k = 100;`
- `char c1 = 'h',  
c2 = 'Y';`
- `bool b = true;`
- `double r1 = 0.0,  
r2 = 1.5E6,  
r3 = 0.56;`



# Sintaxis de declaración de variables

---

- `<declaración> ::=`  
    `<tipo> <declaración-simple>`  
    `{ “,” <declaración-simple> } “;”`
- `<declaración-simple> ::=`  
    `<nombre-variable> [“=” <expresión>]`

# Sintaxis de declaración de variables

---

- `<declaración> ::=`  
    `<tipo> <declaración-simple>`  
    `{ “,” <declaración-simple> } “;”`
- `<declaración-simple> ::=`  
    `<nombre-variable> [“=” <expresión>]`

# Semántica de la declaración de variables

---

- Se reserva espacio en memoria para almacenar tantos datos del tipo especificado en la declaración como `<declaraciones-simples>` haya.
- Si la `<declaración-simple>` de una variable incluye inicialización, se evalúa la `<expresión>` de inicialización. La variable que se declara, pasa a tener ese valor inicial.
- Si la `<declaración-simple>` no incluye una expresión de inicialización, el valor de la variable declarada queda indefinido. Posteriormente en el programa habrá que darle valor inicial antes de consultar su valor.
- A partir de ese punto del programa, se puede trabajar con las variables declaradas (consultar sus valores o modificarlos).

# Declaración de variables

---

## □ Datos de tipos primitivos

- `int a;`
- `int b = 1;`
- `int n = 4 + 8;`
- `char c = char(int('A') + 1);`
- `bool b = (n == 12);`
- `double r = sqrt(2.0);`

# Declaración de variables

---

- ¿Qué valores iniciales tendría cada variable en esta declaración?
  - `int i, j, k = 0;`

# Ejemplo



# Ejemplo

```
#include <iostream>
#include <iomanip>
using namespace std;
/*
 * Programa que escribe en la pantalla la cantidad que
 * equivale en euros a 2000 pesetas.
 */
int main() {
    const double PTAS_POR_EURO = 166.386;
    unsigned pesetas = 2000;
    double euros = pesetas / PTAS_POR_EURO;
    cout << fixed << setprecision(2) << euros << endl;
}
```

# El mismo ejemplo, más general

```
#include <iostream>
#include <iomanip>
using namespace std;
/*
 * Programa que escribe en la pantalla la cantidad
 * equivalente en euros a una cantidad de dinero entera
 * expresada en pesetas solicitada previamente al usuario.
 */
int main() {
    const double PTAS_POR_EURO = 166.386;
    cout << "Escriba una cantidad en pesetas: ";
    unsigned pesetas;
    cin >> pesetas;
    double euros = pesetas / PTAS_POR_EURO;
    cout << fixed << setprecision(2) << euros << endl;
}
```





# Índice

---

- ❑ Datos y tipos de datos
- ❑ Datos primitivos en C++
- ❑ **Expresiones e instrucción de asignación**

# Sintaxis de la instrucción de *asignación*

---

```
<instrucción-asignación> ::=  
    <variable> “=” <expresión> “;”  
    | ...
```

## Semántica de la instrucción de *asignación*

---

- ❑ La `<variable>` de la parte izquierda de la instrucción debe haber sido declarada previamente.
- ❑ Se evalúa la `<expresión>` de la parte derecha.
- ❑ Se modifica el valor de la `<variable>` con el resultado de la evaluación de la `<expresión>`.

# Asignación

```
int m = 3;           // m = 3
int n = m;           // m = 3, n = 3
n = 2 + 7;           // m = 3, n = 9
m = (4 * n) - 2;      // m = 34, n = 9
n = n + 1;           // m = 34, n = 10
```

# Otros operadores de asignación

```
n = n + 1;
```

```
n += 1;
```

```
n++;
```

# Conversión de tipos

---

## □ Tipos

### ■ Respecto a la información

- Conversión sin pérdida de información
- Conversión con pérdida de información

### ■ Respecto a la sintaxis

- Conversión implícita
- Conversión explícita

## Ejemplo

```
#include <iostream>
using namespace std;
/*
 * Programa que comprueba qué conversiones
 * implícitas que realiza C++.
 */
int main() {
    int edad;           cout << edad << endl;
    edad = 18;          cout << edad << endl;
    edad = 17.8;        cout << edad << endl;
    edad = "18";        cout << edad << endl;
    edad = true;        cout << edad << endl;
}
```

# Ejemplo

```
#include <iostream>
using namespace std;
/*
 * Programa que comprueba qué conversiones
 * automáticas que
 */
int main() {
    int edad;
    edad = 18;
    edad = 17.8;
    // edad = "18";
    edad = true;

    cout << edad << endl;
    cout << edad << endl;
    cout << edad << endl;
    cout << edad << endl;
    cout << edad << endl;
}
```

Advertencia:

Se está usando la variable edad, que no está inicializada

Error:

Conversión no válida de **const char\*** (cadena de caracteres) a **int**





# Posible resultado de la ejecución

4201088

18

17

1

## Otro ejemplo más. ¿Qué está mal?

```
#include <iostream>
using namespace std;
/*
 * Programa erróneo que pretende escribir en la
 * pantalla el porcentaje de aprobados
 * correspondiente a 95 estudiantes aprobados con
 * respecto a 160 estudiantes matriculados.
 */
int main() {
    unsigned aprobados = 95;
    unsigned matriculados = 160;

    double porcentaje = aprobados / matriculados * 100;

    cout << porcentaje << endl;
}
```

## ¿Cuáles son correctas?

```
unsigned aprobados = 95;  
unsigned matriculados = 160;
```

```
double tasa;
```

```
tasa = aprobados / matriculados;  
tasa = double(aprobados / matriculados);  
tasa = double(aprobados) / matriculados;  
tasa = aprobados / double(matriculados);  
tasa = double(aprobados) / double(matriculados);
```