

# Fondamenti di programmazione

---

## 3 | Variabili e operatori

# SOMMARIO

- Concetto di variabile
- Cenni sul sistema numerico binario
- Tipi di variabili in C
- I/O - printf e scanf
- Operatori in C

# Variabili e memoria

Un concetto fondamentale alla base di ogni linguaggio di programmazione è quello di **variabile**.

Una variabile è una **locazione di memoria** che contiene un determinato valore

	1	2	3	4	5	6	7	8	...
100									...
200		01110101	10001101	10001110	10101110				...
300									...
400									...
500									...
600									...
700									...
800									...
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

# Variabili e memoria

La memoria può contenere soltanto **bit** e nient'altro!

Un bit è una cifra **binaria**, ovvero un simbolo che può assumere soltanto due valori (**0** oppure **1**)

Un **byte** è una sequenza di **8 bit**

E' essenziale poter rappresentare differenti tipi di variabili, come numeri interi, numeri reali, caratteri . . . .

# Un po' di numeri . . .

Il **sistema numerico binario** è un sistema *numerico posizionale* in base 2.

Differentemente dal sistema numerico decimale in cui vengono utilizzati 10 simboli, nel sistema binario ne vengono utilizzati soltanto due: **1** e **0**.

Un numero binario è una sequenza di cifre binarie (dette appunto **bit**).

Il bit in posizione  $n$  (contando da destra verso sinistra, partendo da 0) va moltiplicata per  $2^n$

La formula per convertire un valore decimale in binario è la seguente:

$$d_n \cdot 2^n + d_{n-1} \cdot 2^{n-1} + \dots + d_1 \cdot 2^1 + d_0 \cdot 2^0 = N_{10}$$

Essendo  $d_i$  il valore del bit  $i$ -esimo

# Un po' di numeri . . .

E' lo stesso principio alla base del sistema *numerico decimale* . . .

Si supponga di considerare il numero **842 317**<sub>10</sub>

Numero decimale	8	4	2	3	1	7
Potenze del 10	$10^5 =$ 100000	$10^4 =$ 10000	$10^3 =$ 1000	$10^2 =$ 100	$10^1 =$ 10	$10^0 =$ 1

Valore decimale  $8 \cdot 10^5 + 4 \cdot 10^4 + 2 \cdot 10^3 + 3 \cdot 10^2 + 1 \cdot 10^1 + 7 \cdot 2^0 = \mathbf{842317}_{10}$

# Un po' di numeri . . .

Allo stesso modo, nel caso di un **numero binario**

$$d_n \cdot 2^n + d_{n-1} \cdot 2^{n-1} + \dots + d_1 \cdot 2^1 + d_0 \cdot 2^0 = N_{10}$$

Si supponga di dover convertire il numero binario **100101**<sub>2</sub> in valore decimale

Numero binario	1	0	0	1	0	1
Potenza di 2	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Valore decimale

$$1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 37_{10}$$

# Un po' di numeri . . .

In genere, è utile conoscere il valore più grande ed il valore più piccolo che si possono rappresentare con  $n$

Con un **byte (8 bit)**:

1. Esistono **256** possibili combinazioni
2. Il numero più grande rappresentabile è **255**

<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
$2^8 = 128$	$2^7 = 64$	$2^6 = 32$	$2^5 = 16$	$2^4 = 8$	$2^3 = 4$	$2^1 = 2$	$2^0 = 1$

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = \mathbf{255}$$



# Un po' di numeri . . .

In genere, è utile conoscere il valore più grande ed il valore più piccolo che si possono rappresentare con  $n$

Con un **byte (8 bit)**:

1. Esistono **256** possibili combinazioni
2. Il numero più grande rappresentabile è **255**

<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
$2^8 = 128$	$2^7 = 64$	$2^6 = 32$	$2^5 = 16$	$2^4 = 8$	$2^3 = 4$	$2^1 = 2$	$2^0 = 1$

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = \mathbf{255}$$

In genere con  $n$  bit

$$\left\{ \begin{array}{l} \text{Valore massimo: } 2^n - 1 \\ \text{Combinazioni: } 2^n \end{array} \right\}$$

# Tipi di variabili in C

Il C prevede solo un piccolo numero di **tipi primitivi** di **variabili**.

Tipo	Rappresenta
char	Caratteri
int	Interi
float	Virgola mobile in <b>singola</b> precisione
double	Virgola mobile in <b>doppia</b> precisione

Esistono **qualificatori** che, se anteposti al tipo, ne modificano le proprietà

• **short**  
• **long**

→ **int**

modificano la **lunghezza**


• **signed**  
• **unsigned**

→ **int**  
**char**

modificano il **segno**

# Tipi di variabili in C

Ma, alla luce di quanto detto, che dimensione hanno i vari tipi??

Tipo	Rappresenta	Dimensione in bit
<b>char</b>	Caratteri	
<b>int</b>	Interi	
<b>float</b>	Virgola mobile in singola precisione	
<b>double</b>	Virgola mobile in doppia precisione	

**Il C non definisce esplicitamente la dimensione dei tipi!**

Le dimensioni dei tipi (int, char, double...) dipendono dall'architettura che si sta utilizzando e dal compilatore.

Dunque, per esempio, la dimensione di un **int** potrebbe variare da un calcolatore all'altro!!

# Tipi di variabili in C

Una variabile viene identificata da un nome che prende, appunto, il nome di **identificatore**.

Un **identificatore**:

1. Può contenere **lettere** (maiuscole e minuscole), **numeri** e **underscore** (trattino basso).
2. Non può iniziare con un numero
3. Non ha limiti di lunghezza
4. Non può coincidere con una **keyword**

Il C è **case sensitive** agli identificatori:

***pip***po      ***pip***Po



***identificatori DIVERSI***

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

**Keyword C**

# Tipi di variabili in C

Una **variabile**, prima di essere utilizzata, deve essere **DICHIARATA** o **DEFINITA**.

**Dichiarazione**

**Definizione**

# Tipi di variabili in C

Una **variabile**, prima di essere utilizzata, deve essere **DICHIARATA** o **DEFINITA**.

## Dichiarazione

Introduce un *identificatore* (ed il tipo).

Indica solamente al compilatore che, quel simbolo, può essere utilizzato nel programma e vi si può far riferimento.

- La stessa variabile può essere dichiarata più volte, anche in punti/file diversi
- La variabile “dichiarata”, ad ogni modo, deve essere stata ***definita*** da qualche parte (altro modulo, altro file...).

## Definizione

# Tipi di variabili in C

Una **variabile**, prima di essere utilizzata, deve essere **DICHIARATA** o **DEFINITA**.

## Dichiarazione

Introduce un *identificatore* (ed il tipo).

Indica solamente al compilatore che, quel simbolo, può essere utilizzato nel programma e vi si può far riferimento.

- La stessa variabile può essere dichiarata più volte, anche in punti/file diversi
- La variabile “dichiarata”, ad ogni modo, deve essere stata ***definita*** da qualche parte (altro modulo, altro file...).

## Definizione

Con la definizione, la variabile viene ***istanziata***, viene cioè *allocata* una porzione di memoria riservata alla variabile.

- Può essere effettuata una sola volta

# Tipi di variabili in C

Una **variabile**, prima di essere utilizzata, deve essere **DICHIARATA** o **DEFINITA**.

## Dichiarazione

Introduce un *identificatore* (ed il tipo).

Indica solamente al compilatore che, quel simbolo, può essere utilizzato nel programma e vi si può far riferimento.

- La stessa variabile può essere dichiarata più volte, anche in punti/file diversi
- La variabile “dichiarata”, ad ogni modo, deve essere stata ***definita*** da qualche parte (altro modulo, altro file...).

## Definizione

Con la definizione, la variabile viene ***istanziata***, viene cioè *allocata* una porzione di memoria riservata alla variabile.

- Può essere effettuata una sola volta



# Tipi di variabili in C

- Definizione semplice

```
tipo    identificatore;
```

- Definizione con **inizializzazione**

```
tipo    id = valore;
```

- Definizione multipla (id1, id2 e id3 saranno variabili dello stesso tipo)

```
tipo    id1, id2, id3;
```

- Definizione multipla con **inizializzazione**

```
tipo    id1 = val1, id2, id3 = val3;
```

# Tipi di variabili in C

Definizione di una variabile:

`tipo    identificatore;`

Esempi di definizioni validi sono:

```
int    pippo = 10;
char   pluto;
float  paperino, paperino2;
long int long_pippo;
short int short_pluto, sh_p2;
unsigned int a__0123__b;
```

Esempi di definizioni **NON** validi sono:

```
int    5_pippo_2;
char   long;
float  pluto-var;
unsigned float myVar;
```

# Inizializzazione di variabili in C

Contestualmente alla **definizione**, è possibile anche **inizializzare** una variabile.

Inizializzare significa **assegnargli** un valore iniziale (infatti, con la sola **definizione**, il valore della variabile è **indefinito**).

```
int v1;
```

## int (solo definizione)

La variabile `v1`, se non viene **inizializzata**, ha valore *indefinito* (cioè potrebbe contenere qualsiasi valore).

```
int v1 = 128;
```

## int

Inizializzazione di una variabile di tipo **int**. Basta specificare il valore

```
char c1 = 'a';
```

## char

Inizializzazione di una variabile di tipo **char**. Il valore da assegnare viene espresso tra **SINGOLI APICI**.

Tale notazione produce il valore *intero* corrispondente alla codifica **ASCII** del carattere indicato.

```
float f1 = 3;
```

```
float f2 = 3.1;
```

```
float f3 = 3.0f;
```

## float e double

Inizializzazione di una variabile di tipo **float**.

# Codifica ASCII

La codifica ASCII prevede l'utilizzo di **7 bit** per la rappresentazione dei caratteri. Come accennato nelle slide precedenti, utilizzando **7 bit** è possibile rappresentare fino a 128 possibili combinazioni (e quindi fino a **128 possibili caratteri**). Soltanto alcuni di questi sono **caratteri stampabili (dal 32 al 126)**.

Valore	Carattere
32	spazio
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

Valore	Carattere
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?
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

Valore	Carattere
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	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g

Valore	Carattere
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	~

# I/O - Funzione printf

La funzione **printf** stampa sullo **stdout** (*standard output*, in genere il monitor) una lista di argomenti coerentemente alla **stringa di formato** specificata.

La stringa di formato ha 2 tipi di *oggetti*:

- caratteri ordinari
- specifiche di conversione (vengono specificati con il simbolo %)

```
printf(stringa_di_formato, lista_argomenti);
```

# I/O - Funzione printf

Esempio di una stringa costante, senza argomenti

```
printf("Hello world\n");
```

- Non sono presenti **argomenti**
- Viene stampata a schermo la stringa "Hello world" (\n, carattere di *new line*)

Sequenza di escape	Descrizione
<code>\n</code>	Carattere di <i>new line</i> . Inserisce un ritorno a capo
<code>\t</code>	Inserisce un carattere di tabulazione
<code>\\</code>	Inserisce il carattere <i>backslash</i>
<code>\"</code>	Inserisce i <i>doppi apici</i> (evita che vengano intesi come fine stringa)

# I/O - Funzione printf

Le **specificazioni di conversione**, che vanno inserite dentro la stringa di formato utilizzando il simbolo %, indicano alla **printf** di stampare, nello **stdout**, il valore di un argomento, specificato nella **lista di argomenti**.

```
int v1 = 10, v2 = 15, v3 = 20;  
printf("Valori = %d, %d, %d\n", v1, v2, v3);
```

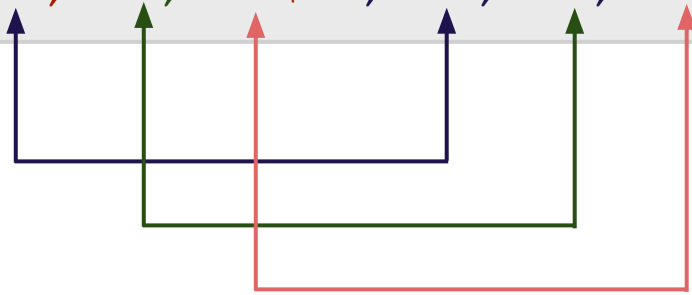
Stringa di formato

Argomenti

# I/O - Funzione printf

Le **specificazioni di conversione**, che vanno inserite dentro la stringa di formato utilizzando il simbolo %, indicano alla **printf** di stampare, nello **stdout**, il valore di un argomento, specificato nella **lista di argomenti**.

```
int v1 = 10, v2 = 15, v3 = 20;  
printf("Valori = %d, %d, %d\n", v1, v2, v3);
```



The diagram illustrates the mapping between the format specifiers in the printf function call and the variables in the argument list. Three colored arrows (purple, green, and red) originate from the format specifiers %d, %d, and %d in the string "Valori = %d, %d, %d\n" and point to the variables v1, v2, and v3 respectively. The arrows are color-coded to match the corresponding variable: purple for v1, green for v2, and red for v3.

Stampa a schermo la stringa

*"Valori = 10, 15, 20"*



# I/O - Funzione printf

In base al **tipo di variabile** che si intende stampare, bisogna utilizzare la giusta **specificazione di conversione**.

Tipo	Specificatore di tipo
%c	Caratteri
%d (o %i)	Interi
%f	Reali (double e float)
%s	Stringhe
%x - %X	Notazione esadecimale
%o	Notazione ottale
%e	Notazione scientifica
-----	
%p	Puntatori

```
int v1 = 6;  
printf("v1 = %d\n", v1);
```

```
char c1 = 'f';  
printf("c1 = %c\n", c1);
```

```
float f1 = 3.14;  
printf("f1 = %f\n", f1);
```

# Operatore unario *di indirizzo*

Prima di introdurre la funzione **scanf** (duale alla printf e che permette l'acquisizione di dati) è bene introdurre l'**operatore unario di indirizzo**.

Come introdotto nelle slide precedenti, una variabile è una **porzione di memoria** atta a contenere dati.

	+	1	2	3	4	5	6	7	8	...
100										...
200		(int) pippo = 17;								...
300										...
400				(int) pluto = 28;						...
500										...
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.

# Operatore unario *di indirizzo*

Prima di introdurre la funzione **scanf** (duale alla printf e che permette l'acquisizione di dati) è bene introdurre l'**operatore unario di indirizzo**.

Come introdotto nelle slide precedenti, una variabile è una **porzione di memoria** atta a contenere dati.

	+	1	2	3	4	5	6	7	8	...
100										...
200		(int) pippo = 17;								...
300										...
400				(int) pluto = 28;						...
500										...
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.

pippo → 17

pluto → 28

# Operatore unario *di indirizzo*

Prima di introdurre la funzione **scanf** (duale alla printf e che permette l'acquisizione di dati) è bene introdurre l'**operatore unario di indirizzo**.

Come introdotto nelle slide precedenti, una variabile è una **porzione di memoria** atta a contenere dati.

+	1	2	3	4	5	6	7	8	...
100									...
200		(int) pippo = 17;							...
300									...
400				(int) pluto = 28;					...
500									...
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

pippo → 17

&pippo → 0x202

pluto → 28

&pluto → 0x404

# I/O - Funzione scanf

Se la funzione `printf` permette di inviare dati nello ***stdout***, allo stesso modo, la funzione ***scanf*** permette di acquisire dati dallo ***standard input*** (*stdin*, in genere la tastiera).

`scanf(stringa_di_formato, argomenti)`

- Nella stringa di formato, bisogna indicare come devono essere “trattati” i dati letti dallo standard input.
- Gli argomenti, invece, rappresentano le variabili nelle quali memorizzare i valori letti e convertiti.
- Va specificato l'indirizzo di memoria della variabile

# I/O - Funzione scanf

Se la funzione `printf` permette di inviare dati nello ***stdout***, allo stesso modo, la funzione **`scanf`** permette di acquisire dati dallo ***standard input*** (*stdin*, in genere la tastiera).

```
int v1, v2;  
scanf( "%d%d", &v1, &v2 );
```

Stringa di formato

Argomenti

Nell'esempio, si intende acquisire due interi.

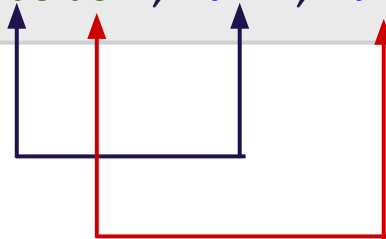
Il programma resterà in attesa che l'utente inserisca due valori interi.

Il simbolo **`&`** che precede il nome delle variabili è un **operatore unario** (ha un solo operando) che restituisce l'indirizzo di memoria dell'operando a cui è applicato.

# I/O - Funzione scanf

Se la funzione printf permette di inviare dati nello **stdout**, allo stesso modo, la funzione **scanf** permette di acquisire dati dallo **standard input** (*stdin*, in genere la tastiera).

```
int v1, v2;  
scanf( "%d%d", &v1, &v2 );
```



Come nella printf, nella stringa di formato compaiono le **specifiche di conversione** (simboli preceduti dal %).

Per ogni “%” deve essere presente un argomento

# Operatori

In C esistono differenti tipologie di operatori:

- Operatori **aritmetici**
- Operatori **relazionali**
- Operatori **logici**
- Operatori **bit-a-bit**

## Cardinalità di un operatore

Numero di operandi che richiede  
(operatori *unari*, *binari*, *ternari*)

In realtà sono definiti altri operatori, fra i quali il più *importante* è senz'altro l'**operatore di assegnamento**.

Viene usato per assegnare, ad una variabile, un valore.

```
int v1 = 69;  
char c = 'a';
```



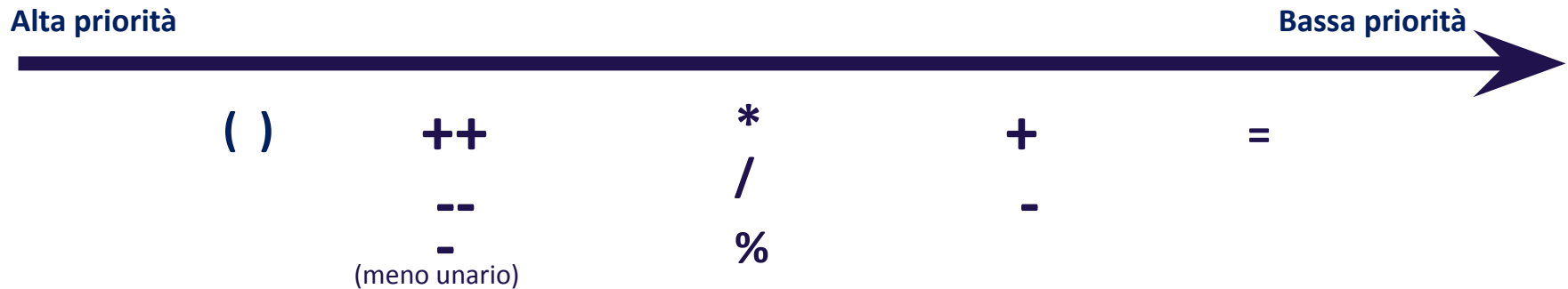
# Operatori Aritmetici

Gli operatori aritmetici sono mostrati nella tabella seguente e permettono di effettuare operazioni aritmetiche fra valori e variabili.

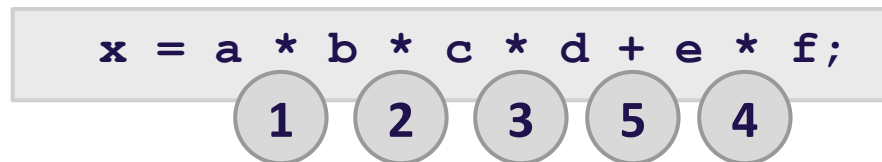
Operatore	Azione	Cardinalità
-	Sottrazione (o meno <b>unario</b> )	Binario (unario)
+	Addizione	Binario
*	Moltiplicazione	Binario
/	Divisione	Binario
%	Modulo	Binario
--	Decremento unitario	Unario
++	Incremento unitario	Unario
=	Assegnamento	Binario
( )	Modifica ordine di valutazione	...

# Operatori Aritmetici - priorità

Le **espressioni** vengono **valutate** secondo le **priorità** associate a ciascun operatore.



Se gli operatori sono *nello stesso livello*, vengono eseguite **da sinistra a destra**



# Operatori Aritmetici - forma compatta

Gli **operatori aritmetici** possono essere combinati con l'**operatore di assegnamento** in forme più compatte. Per esempio, un'operazione di assegnamento della seguente forma:

```
int v1 = 10;  
v1 = v1 + 2;
```

Può essere scritta in maniera più compatta e sintetica utilizzando la notazione

```
int v1 = 10;  
v1 += 2;
```

Tale forma può essere utilizzata con tutti gli operatori aritmetici (+ \* / - ...)

```
int v1 = 10;  
v1 -= 2;
```

```
int v1 = 10;  
v1 *= 2;
```

```
int v1 = 10;  
v1 /= 2;
```

# Operatori Aritmetici - incremento e decremento

Gli operatori di **incremento** e **decremento** possono essere utilizzati in modo **pre-fisso** o **post-fisso**.

```
int v1 = 10;  
++v1;
```

**Pre-fisso**

```
int v1 = 10;  
v1++;
```

**Post-fisso**

Nella notazione **pre-fissa** la variabile viene **prima incrementata** e poi valutata.

```
int v1 = 10;
```

```
printf("%d", ++v1);
```

Stampa **11**

```
printf("%d", v1);
```

Stampa **11**

Nella notazione **post-fissa** la variabile viene **prima valutata** e solo successivamente incrementata

```
int v1 = 10;
```

```
printf("%d", v1++);
```

Stampa **10**

```
printf("%d", v1);
```

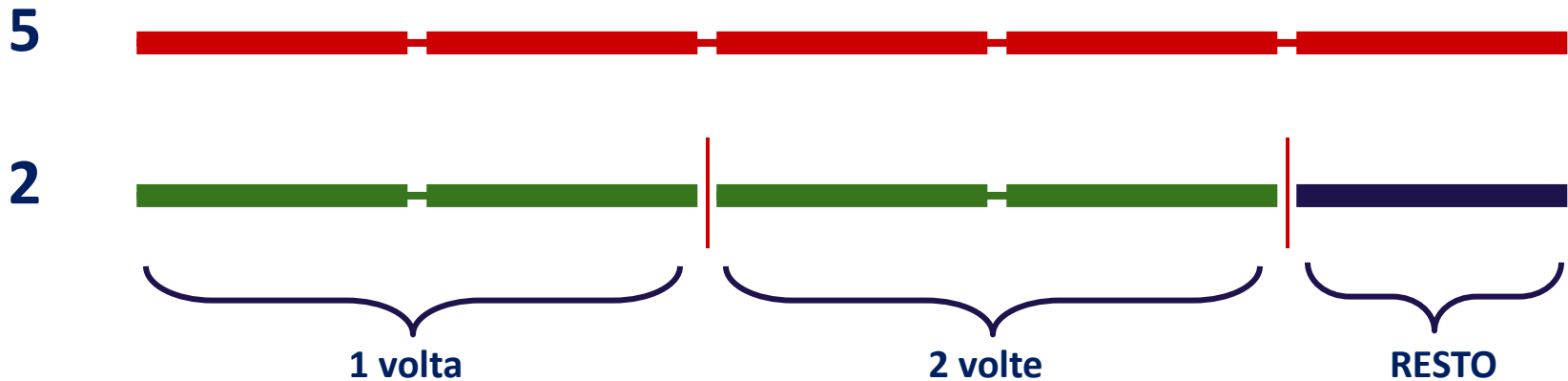
Stampa **11**

# Operatori Aritmetici - operatore modulo

L'operatore binario % (**modulo**), che si applica a due interi, restituisce il **resto** della **divisione intera**.

Esempio:

$5 \% 2$



# Operatori Aritmetici - operatore modulo

L'operatore binario % (**modulo**), che si applica a due interi, restituisce il **resto** della **divisione intera**.

Operazione	Risultato Divisione	Resto
5 / 2	2 (2 × 2 = 4)	1 (5 - 2 × 2 = 1)
13 / 3	4 (3 × 4 = 12)	1 (13 - 3 × 4 = 1)
7 / 4	1 (4 × 1 = 4)	3 (7 - 4 × 1 = 3)
10 / 5	2 (5 × 2 = 10)	0 (10 - 5 × 2 = 0)
4 / 7	0 (7 × 0 = 0)	4 (4 - 7 × 0 = 4)
...	...	...

```
int v1=5, v2=2, v3=18, v4;
```

```
v4 = v1 % v2;
```

La variabile **v4** avrà valore **1**

```
v4 = v3 % v1;
```

La variabile **v4** avrà valore **3**

```
v4 = v3 % v2;
```

La variabile **v4** avrà valore **0**

# Operatori Relazionali

Gli **operatori relazionali** servono per, appunto, *stabilire in che relazione* sono due valori.

Le espressioni che coinvolgono gli operatori relazionali **producono** dei valori **booleani** (valori logici, vero o falso).

Operatore	Azione	Es. VERO	Es. FALSO
<b>==</b>	<b>VERO</b> se i due valori sono <b>uguali</b>	<b>18 == 18</b>	<b>30 == 18</b>
<b>!=</b>	<b>VERO</b> se i due valori sono <b>diversi</b>	<b>15 != 12</b>	<b>17 != 17</b>
<b>&gt;</b>	<b>VERO</b> se il primo operando è <b>strettamente maggiore</b> del secondo	<b>20 &gt; 19</b>	<b>21 &gt; 21</b>
<b>&lt;</b>	<b>VERO</b> se il primo operando è <b>strettamente minore</b> del secondo	<b>12 &lt; 13</b>	<b>13 &lt; 8</b>
<b>&gt;=</b>	<b>VERO</b> se il primo operando è <b>maggiore o uguale</b> del secondo	<b>23 &gt;= 23</b>	<b>23 &gt;= 16</b>
<b>&lt;=</b>	<b>VERO</b> se il primo operando è <b>minore o uguale</b> del secondo	<b>12 &lt;= 12</b>	<b>4 &lt;= 3</b>

In genere, il valore **booleano** ritornato dalla valutazione di un'espressione, è rappresentato da un **int**.

# Operatori Relazionali

In genere, il valore **booleano** ritornato dalla valutazione di un'espressione, è rappresentato da un **int**.

```
int v1 = 10, v2 = 5;  
printf("Val = %d", v1 == v2);
```

10 non è uguale a 5. Espressione **FALSA**  
**Val = 0**

```
int v1 = 10, v2 = 10;  
printf("Val = %d", v1 == v2);
```

10 è uguale a 10. Espressione **VERA**  
**Val = 1**

```
int v1 = 10, v2 = 17;  
printf("Val = %d", v1 > v2);
```

10 non è maggiore di 17.  
Espressione **FALSA**.  
**Val = 0**



# Operatori Logici

Gli **operatori logici** permettono di combinare più **espressioni logiche** (che producono cioè un valore **booleano**), producendo ancora un valore logico.

Operatore logico	Simbolo in C	Significato
<b>AND</b>	<b>&amp;&amp;</b>	<b>VERO</b> se <b>ENTRAMBI</b> gli operandi hanno valore <b>VERO</b>
<b>OR</b>	<b>  </b>	<b>VERO</b> se <b>ALMENO UNO</b> degli operandi ha valore <b>VERO</b>
<b>NOT</b>	<b>!</b>	<b>NEGA</b> il valore logico cui è applicato
<b>XOR</b>	<b>^</b>	<b>VERO</b> se, e solo se, <b>UNO SOLO</b> degli operandi ha valore <b>VERO</b>

```
int v1 = 10;  
if (v1 > 0 && v1 < 20) { ... }
```

**Espressione 1**  
(produce un  
valore logico,  
**operando 1**)

**Espressione 2**  
(produce un  
valore logico,  
**operando 2**)

```
int v1 = 10;  
if ( ! (v1 == 10) ) { ... }
```

**Espressione**  
(produce un valore  
logico, **operando**)

# Altri operatori

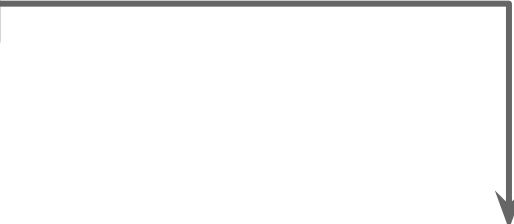
Oltre a quelli presentati, il C definisce anche altri operatori. Fra questi, quelli che vengono utilizzati più spesso sono:

- **?** (operatore ternario)
- **&** (operatore unario di *indirizzo*)
- **\*** (operatore di *dereferenziazione*)
- **sizeof**
- **.** (operatore *punto*)
- **->** (operatore *freccia*)

# Altri operatori

Oltre a quelli presentati, il C definisce anche altri operatori. Fra questi, quelli che vengono utilizzati più spesso sono:

- **?** (operatore ternario)
- **&** (operatore unario di *indirizzo*)
- **\*** (operatore di *dereferenziazione*)
- **sizeof**
- **.** (operatore *punto*)
- **->** (operatore *freccia*)



E' un operatore **ternario** (richiede cioè **3 operandi**).  
Può essere considerato come una scrittura più compatta del  
costrutto **if else**

`espressione1 ? espressione2 : espressione3`

Se **espressione1** è **VERA** viene eseguita

**espressione2**, altrimenti viene eseguita **espressione3**

# Altri operatori

Oltre a quelli presentati, il C definisce anche altri operatori. Fra questi, quelli che vengono utilizzati più spesso sono:

- ? (operatore ternario)
- **&** (operatore unario di **indirizzo**)
- **\*** (operatore di **dereferenziazione**)
- sizeof
- . (operatore **punto**)
- -> (operatore **freccia**)

Sono operatori **unari** (richiedono cioè un solo operando) ed entrano in gioco quando si inizia a parlare di **puntatori**.

L'operatore **&**, applicato ad una variabile, ne restituisce l'indirizzo di memoria.

L'operatore **\*** (duale all'operatore **&**) è detto di **dereferenziazione** e permette di accedere al contenuto partendo dalla conoscenza dell'indirizzo di memoria.

**VERRANNO DISCUSSI NEL  
SEGUITO DELLE LEZIONI  
SUCCESSIVE!!**

# Altri operatori

Oltre a quelli presentati, il C definisce anche altri operatori. Fra questi, quelli che vengono utilizzati più spesso sono:

- ? (operatore ternario)
- & (operatore unario di *indirizzo*)
- \* (operatore di *dereferenziazione*)
- **sizeof**
- . (operatore *punto*)
- -> (operatore *freccia*)

L'operatore **sizeof** viene utilizzato per calcolare la dimensione di un **tipo di variabile**.

Come mostrato nelle slide precedenti, il C non definisce esplicitamente le dimensioni dei tipi ma dipende dall'architettura e dal compilatore. Per conoscere la dimensione di un **tipo di variabile** (che sia `int`, `float` o altro), si usa l'operatore **sizeof**. Si usa nel seguente modo:

Modi  
equivalenti

```
int size_char;
char var_char;
size_char = sizeof(char);
size_char = sizeof(var_char);
size_char = sizeof var_char;
```

Dunque, supponendo di voler conoscere la dimensione del tipo **char**, possiamo indicare:

- direttamente il **tipo char**
- **una variabile** di tipo char

# Altri operatori

Oltre a quelli presentati, il C definisce anche altri operatori. Fra questi, quelli che vengono utilizzati più spesso sono:

- ? (operatore ternario)
- & (operatore unario di *indirizzo*)
- \* (operatore di *dereferenziazione*)
- sizeof

- . (operatore *punto*)
- -> (operatore *freccia*)

Operatori che servono per la manipolazione di **strutture** (tipi di dato definiti dall'utente).

**VERRANNO DISCUSSI NEL  
SEGUITO DELLE LEZIONI  
SUCCESSIVE!!**