

# LINGUAGGI DI PROGRAMMAZIONE

---

Il “potere espressivo” di un linguaggio è caratterizzato da:

- **quali tipi di dati** consente di rappresentare (direttamente o tramite definizione dell'utente)
- **quali istruzioni di controllo** mette a disposizione (quali operazioni e in quale ordine di esecuzione)

***PROGRAMMA = DATI + CONTROLLO***

# IL LINGUAGGIO C

---

## UN PO' DI STORIA

- definito nel 1972 (AT&T Bell Labs) per sostituire l'Assembler
- prima definizione precisa: Kernigham & Ritchie (1978)
- prima definizione ufficiale: ANSI (1983)

# IL LINGUAGGIO C

---

## CARATTERISTICHE

- linguaggio *sequenziale, imperativo, strutturato* a blocchi
- usabile anche come linguaggio di sistema
  - adatto a software di base, sistemi operativi, compilatori, ecc.
- portabile, efficiente, sintetico
  - ma a volte poco leggibile...

# IL LINGUAGGIO C

---

## **Basato su pochi *concetti elementari***

- dati (tipi primitivi, tipi di dato)
- espressioni
- dichiarazioni / definizioni
- funzioni
- istruzioni / blocchi

# IL LINGUAGGIO C

---

- Un elaboratore è un **manipolatore di simboli (segni)**
- L'architettura fisica di ogni elaboratore è **intrinsecamente capace** di trattare vari domini di dati, detti **tipi primitivi**
  - dominio dei **numeri naturali e interi**
  - dominio dei **numeri reali**  
(con qualche approssimazione)
  - dominio dei **caratteri**
  - dominio delle **stringhe di caratteri**

# TIPI DI DATO

---

Il concetto di *tipo di dato* viene introdotto per raggiungere due obiettivi:

- esprimere in modo sintetico
  - la loro rappresentazione in memoria, e
  - un insieme di operazioni ammissibili
- permettere di *effettuare controlli statici* (al momento della compilazione) sulla *correttezza* del programma.

# TIPI DI DATO PRIMITIVI IN C

---

- **caratteri**
  - `char`            caratteri ASCII
  - `unsigned char`

# TIPI DI DATO PRIMITIVI IN C

---

- **caratteri**

- `char`                    caratteri ASCII
- `unsigned char`

- **interi con segno**

- `short (int)` -32768 ... 32767 (16 bit)
- `int`                        ????????
- `long (int)` -2147483648 .... 2147483647 (32 bit)

Dimensione di `int`  
e `unsigned int`  
non fissa. **Dipende**  
**dal compilatore**



# TIPI DI DATO PRIMITIVI IN C

---

- **caratteri**

- `char`                    caratteri ASCII
- `unsigned char`

Dimensione di `int`  
e `unsigned int`  
non fissa. **Dipende**  
**dal compilatore**

- **interi con segno**

- `short (int)` -32768 ... 32767 (16 bit)
- `int`                        ????????
- `long (int)` -2147483648 .... 2147483647 (32 bit)

- **naturali (interi senza segno)**

- `unsigned short (int)` 0 ... 65535 (16 bit)
- `unsigned (int)`                        ????????
- `unsigned long (int)` 0 ... 4294967295 (32 bit)

# TIPI DI DATO PRIMITIVI IN C

---

- **reali**
  - **float**                      singola precisione (32 bit)
  - **double**                    doppia precisione (64 bit)

# TIPI DI DATO PRIMITIVI IN C

---

- **reali**
  - `float`                      singola precisione (32 bit)
  - `double`                      doppia precisione (64 bit)
  
- **boolean**
  - ***non esistono in C come tipo a sé stante***
  - si usano gli interi:
    - **zero** indica **FALSO**
    - ogni altro valore indica **VERO**
  - convenzione: suggerito utilizzare **uno** per **VERO**

# COSTANTI DI TIPI PRIMITIVI

---

- **interi** (in varie basi di rappresentazione)

<i>base</i>	<i>2 byte</i>	<i>4 byte</i>
decimale	<b>12</b>	<b>70000, 12L</b>
ottale	<b>014</b>	<b>0210560</b>
esadecimale	<b>0xFF</b>	<b>0x11170</b>

# COSTANTI DI TIPI PRIMITIVI

---

- **interi** (in varie basi di rappresentazione)

<i>base</i>	<i>2 byte</i>	<i>4 byte</i>
decimale	<b>12</b>	<b>70000, 12L</b>
ottale	<b>014</b>	<b>0210560</b>
esadecimale	<b>0xFF</b>	<b>0x11170</b>

- **reali**

– in doppia precisione

**24.0      2.4E1      240.0E-1**

– in singola precisione

**24.0F      2.4E1F      240.0E-1F**

# COSTANTI DI TIPI PRIMITIVI

---

- **caratteri**

- singolo carattere racchiuso fra apici

`'A'`    `'C'`    `'6'`

- caratteri speciali:

`'\n'`    `'\t'`    `'\''`    `'\\'`    `'\"'`

# STRINGHE

---

- Una *stringa* è una *sequenza di caratteri* delimitata da virgolette

"ciao"

"Hello\n"

- In C le stringhe sono semplici sequenze di caratteri di cui l'ultimo, *sempre presente in modo implicito*, è '\0'

"ciao" = {'c', 'i', 'a', 'o', '\0'}

# ESPRESSIONI

---

- Il C è un linguaggio basato su *espressioni*
- Una *espressione* è una *notazione che denota un valore* mediante un processo di *valutazione*
- Una espressione può essere *semplice* o *composta* (tramite aggregazione di altre espressioni)



# ESPRESSIONI SEMPLICI

---

## Quali espressioni elementari?

- **costanti**

- ‘A’ 23.4 -3 “ciao” ....

- **simboli di variabile**

- x pippo pigreco ....

- **simboli di funzione**

- $f(x)$

- `concat("alfa", "beta")`

- ...

# OPERATORI ED ESPRESSIONI COMPOSTE

---

- Ogni linguaggio introduce un **insieme di operatori**
- che permettono di **aggregare altre espressioni (operandi)**
- per formare **espressioni composte**
- con riferimento a diversi **domini / tipi di dato** (numeri, testi, ecc.)

## Esempi

`2 + f(x)`

`4 * 8 - 3 % 2 + arcsin(0.5)`

`strlen(strcat(Buf, "alfa"))`

`a && (b || c)`

`...`

# CLASSIFICAZIONE DEGLI OPERATORI

---

- **Due criteri di classificazione:**
  - in base al *tipo* degli operandi
  - in base al *numero* degli operandi

in base al <i>tipo</i> degli operandi	in base al <i>numero</i> di operandi
<ul style="list-style-type: none"><li>• aritmetici</li><li>• relazionali</li><li>• logici</li><li>• condizionali</li><li>• ...</li></ul>	<ul style="list-style-type: none"><li>• unari</li><li>• binari</li><li>• ternari</li><li>• ...</li></ul>

# OPERATORI ARITMETICI

---

<i>operazione</i>	<i>operatore</i>	<b>C</b>
inversione di segno	<i>unario</i>	-
somma	<i>binario</i>	+
differenza	<i>binario</i>	-
moltiplicazione	<i>binario</i>	*
divisione fra interi	<i>binario</i>	/
divisione fra reali	<i>binario</i>	/
modulo (fra interi)	<i>binario</i>	%

**NB:** la divisione  $a/b$  è fra interi se sia  $a$  sia  $b$  sono interi, è fra reali in tutti gli altri casi

# OPERATORI: OVERLOADING

---

- In C (come in Pascal, Fortran e molti altri linguaggi) operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo (ad esempio, le operazioni aritmetiche su reali o interi).
- In realtà l'operazione è diversa e può produrre risultati diversi.

```
int X,Y;  
se X = 10 e Y = 4;  
X/Y vale 2
```

```
int X; float Y;  
se X = 10 e Y = 4.0;  
X/Y vale 2.5
```

```
float X,Y;  
se X = 10.0 e Y = 4.0;  
X/Y vale 2.5
```

# CONVERSIONI DI TIPO

---

- In C è possibile combinare tra di loro operandi di tipo diverso:
  - espressioni **omogenee**: tutti gli operandi sono dello stesso tipo
  - espressioni **eterogenee**: gli operandi sono di tipi diversi.
- **Regola adottata in C:**
  - sono eseguibili le espressioni eterogenee in cui tutti i tipi referenziati risultano **compatibili** (cioè: dopo l'applicazione della regola automatica di conversione implicita di tipo del C risultano omogenei ).

# CONVERSIONI DI TIPO

---

- Data una espressione  $x \text{ op } y$ .
  - 1. Ogni variabile di tipo **char** o **short** viene convertita nel tipo **int**;
  - 2. Se dopo l'esecuzione del passo 1 l'espressione è ancora eterogenea, rispetto alla seguente gerarchia  
 $\text{int} < \text{long} < \text{float} < \text{double} < \text{long double}$   
si converte temporaneamente l'operando di tipo *inferiore* al tipo *superiore* (***promotion***);
  - 3. A questo punto l'espressione è **omogenea** e viene eseguita l'operazione specificata. Il risultato è di tipo uguale a quello prodotto dall'operatore effettivamente eseguito. (In caso di overloading, quello più alto gerarchicamente).

# CONVERSIONI DI TIPO

---

```
int x;  
char y;  
double r;  
(x+y) / r
```



La valutazione dell'espressione procede da sinistra verso destra

- **Passo 1:**  $(x+y)$ 
  - $y$  viene convertito nell'intero corrispondente
  - viene applicata la somma tra interi
  - **risultato intero**  $tmp$
- **Passo 2**
  - $tmp / r$   $tmp$  viene convertito nel double corrispondente
  - viene applicata la divisione tra reali
  - **risultato reale**



# OPERATORI RELAZIONALI

---

Sono tutti operatori *binari*:

<i>relazione</i>	<b>C</b>
<b>uguaglianza</b>	<b>==</b>
<b>diversità</b>	<b>!=</b>
<b>maggiore di</b>	<b>&gt;</b>
<b>minore di</b>	<b>&lt;</b>
<b>maggiore o uguale a</b>	<b>&gt;=</b>
<b>minore o uguale a</b>	<b>&lt;=</b>

# OPERATORI RELAZIONALI

---

## Attenzione:

- non esistendo il tipo *boolean*, in C le espressioni relazionali *denotano un valore intero*
  - 0 denota *falso*  
(condizione non verificata)
  - 1 denota *vero*  
(condizione verificata)

# OPERATORI LOGICI

---

<i>connettivo logico</i>	<i>operatore</i>	<i>C</i>
not (negazione)	<i>unario</i>	<b>!</b>
and	<i>binario</i>	<b>&amp;&amp;</b>
or	<i>binario</i>	<b>  </b>

- Anche **le espressioni logiche denotano un valore intero**
- da interpretare come vero (1) o falso (0)

# PRIORITA' DEGLI OPERATORI

---

- **PRIORITÀ:** specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori (infissi) diversi*
- **Esempio:**  $3 + 10 * 20$ 
  - si legge come  $3 + (10 * 20)$  perché l'operatore  $*$  è più prioritario di  $+$
- **NB:** operatori diversi possono comunque avere *egual priorità*

# ASSOCIATIVITA' DEGLI OPERATORI

---

- **ASSOCIATIVITÀ:** specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori (infissi) di egual priorità*
- Un operatore può quindi essere *associativo a sinistra* o *associativo a destra*
- **Esempio:** **3 - 10 + 8**
  - si legge come  $(3 - 10) + 8$  perché gli operatori - e + sono equiprioritari e **associativi a sinistra**

# PRIORITA' e ASSOCIATIVITA'

---

- **Priorità e associatività predefinite possono essere alterate mediante *l'uso di parentesi***
- **Esempio:**  $(3 + 10) * 20$ 
  - denota 260 (anziché 203)
- **Esempio:**  $30 - (10 + 8)$ 
  - denota 12 (anziché 28)

# RIASSUNTO OPERATORI DEL C

---

Priorità	Operatore	Simbolo	Associatività
1 (max)	chiamate a funzione selezioni	() [] -> .	a sinistra
2	operatori unari: op. negazione op. aritmetici unari op. incr. / decr. op. indir. e deref. op. sizeof	! + ++ & sizeof	a destra
3	op. moltiplicativi	* / %	a sinistra
4	op. additivi	+ -	a sinistra

# RIASSUNTO OPERATORI DEL C

Priorità	Operatore	Simbolo	Associatività
5	op. di shift	>> <<	a sinistra
6	op. relazionali	< <= > >=	a sinistra
7	op. uguaglianza	== !=	a sinistra
8	op. di AND bit a bit	&	a sinistra
9	op. di XOR bit a bit	^	a sinistra
10	op. di OR bit a bit		a sinistra
11	op. di AND logico	&&	a sinistra
12	op. di OR logico		a sinistra
13	op. condizionale	? . . . :	a destra
14	op. assegnamento e sue varianti	= += -= *= /= %= &= ^=  = <<= >>=	a destra
15 (min)	op. concatenazione	,	a sinistra