

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`

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

- **interi con segno**

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

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` ???????? (di solito 32...)
- `long (int)` -2147483648 2147483647 (32 bit)

- **naturali (interi senza segno)**

- `unsigned short (int)` 0 ... 65535 (16 bit)
- `unsigned (int)` ???????? (di solito 32...)
- `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	12	70000
ottale	014	0210560
esadecimale	0xC	0x11170

COSTANTI DI TIPI PRIMITIVI

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

<i>base</i>	<i>2 byte</i>	<i>4 byte</i>
decimale	12	70000
ottale	014	0210560
esadecimale	0xC	0x11170

- **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">• aritmetici• relazionali• logici• condizionali• ...	<ul style="list-style-type: none">• unari• binari• ternari• ...

OPERATORI ARITMETICI

<i>operazione</i>	<i>operatore</i>	C
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>	C
uguaglianza	==
diversità	!=
maggiore di	>
minore di	<
maggiore o uguale a	>=
minore o uguale a	<=

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>	!
and	<i>binario</i>	&&
or	<i>binario</i>	

- 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

TABELLA ASCII

Codice ASCII:

La tabella ASCII (American Standard Code for Information Interchange) è un codice convenzionale usato rappresentare i caratteri di testo attraverso i byte: ad ogni byte viene fatto corrispondere un diverso carattere della tastiera (lettere, numeri, segni).

ASCII standard: 7 bit

(→ 128 caratteri)

ASCII estesa: 8 bit (1 byte)

(→ 256 caratteri)

Dec	Sym	Dec	Char	Dec	Char	Dec	Char
0	NUL	32		64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	TAB	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	␣

Tabella ASCII Standard

Dec	Char	Dec	Char	Dec	Char	Dec	Char
128	Ç	160	á	192	+	224	Ó
129	ü	161	í	193	-	225	ß
130	é	162	ó	194	-	226	Ô
131	â	163	ú	195	+	227	Ò
132	ä	164	ñ	196	-	228	ö
133	à	165	Ñ	197	+	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	+	232	ƒ
137	ë	169	®	201	+	233	Û
138	è	170	¬	202	-	234	Ü
139	ï	171	½	203	-	235	Ù
140	î	172	¼	204	¡	236	Ý
141	ì	173	¡	205	-	237	Ÿ
142	Ä	174	«	206	+	238	–
143	Å	175	»	207	º	239	ˆ
144	É	176	–	208	¶	240	Û
145	æ	177	–	209	Ð	241	±
146	Æ	178	–	210	Ê	242	–
147	ô	179	¡	211	Ë	243	¾
148	õ	180	¡	212	Ë	244	¶
149	ò	181	À	213	ì	245	§
150	û	182	Â	214	í	246	÷
151	ù	183	À	215	î	247	ˆ
152	ÿ	184	©	216	ï	248	°
153	Ö	185	¡	217	+	249	–
154	Û	186	¡	218	+	250	ˆ
155	ø	187	+	219	–	251	ˆ
156	£	188	+	220	–	252	ˆ
157	Ø	189	¢	221	¡	253	ˆ
158	×	190	¥	222	¡	254	–
159	ƒ	191	+	223	–	255	–

Tabella ASCII estesa