ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Main e Blocchi

Struttura di un Programma in C

La struttura di un programma in C è:

```
<programma> :: =
        {<unità di traduzione>}
        <main>
        {<unità di traduzione>}
```

- Il programma è una serie di "unità di traduzione"
  - Si ricorda che {} indica "zero o più"
- Una di queste si chiama sempre main

Main

In prima approssimazione, il main ha come sintassi:

```
<main> :: =
       <tipo> main() <blocco>

<blocco> ::= { {<istruzione>} }
```

Il main è il punto di partenza del programma:

- Le parti in rosso fanno parte della sintassi
- La parola chiave "main" e le parentesi compaiono sempre
- `<tipo>` è un tipo primitivo (sempre `int` o `void`)

Il main è in realtà una funzione: ne parleremo più avanti

Main

In prima approssimazione, il main ha come sintassi:

```
<main> :: =
       <tipo> main() <blocco>

<blocco> ::= { {<istruzione>} }
```

Un blocco è una sequenza di istruzioni

- È sempre racchiuso tra parentesi graffe
- Ogni istruzione "dice" all'elaboratore di fare qualcosa
- Le istruzioni sono eseguite nell'ordine in cui appaiono

Main: Esempio

Un programma con main vuoto:

```
int main() {
        return 0;
}
```

Il programma è corretto, esegue, ma non fa nulla

- "`return 0;`" termina il programma...
- ...e notifica che non ci sono stati errori
- Anche dell'istruzione "return" riparleremo più avanti

Main: Esempio

Un programma con main vuoto:

```
int main() {
        return 0;
}
```

Codifica per il testo

- Il C usa la codifica ASCII (ne riparleremo tra poco)
- In sintesi: solo caratteri inglesi!
- Niente lettere accentate, etc.

Indentazione

Un programma con main vuoto:

```
int main() {
        return 0;
}
```

Si chiama indentazione la spaziatura all'inizio di una riga

- Il compilatore la trascura: non è necessaria
- Ma per un umano è molto utile!
- Evidenzia quali istruzioni facciano parte di un blocco

Commenti

Un programma con main vuoto:

```
int main() {
        /* un commento */
        return 0; // commento fino a fine riga
}
```

Il testo tra /* e */ è un commento

- Il compilatore lo ignora, ma per un umano è molto utile!
  - Comunica le nostre intenzioni, funge da promemoria
- Sintassi alternativa: //
  - Viene ignorato il testo fino a fine riga

Commenti

I commenti sono preziosi!

- Un giorno un programmatore rimise mano a del vecchio codice
- …E vi trovò questo commento:

```
/*
  When I wrote this, only God and I knew what I was doing.
  Now, God only knows.
*/
```

Quali Istruzioni?

Nelle prossime lezoni vedremo quali istruzion sono supportate

In questo blocco di slide ne vedremo due tipi:

- Espressioni
- Definizioni di variabili

Sufficienti per usare il nostro PC come... 😉

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Tipi di Dato in C

Linguaggi e Costrutti Elementari

Un elaboratore è una entità che può:

- Memorizzare informazioni
- Eseguire su di esse alcune operazioni elementari

Abbiamo anche notato che:

- Un linguaggio di programmazione di alto livello…
- …È basato su una astrazione di elaboratore

Quindi anche per il C dobbiamo definire questi due aspetti:

- Quali informazioni siano memorizzabili e manipolabili
- Quali operazioni possiamo effettuare

Tipi di Dato

Le informazioni manipolabili sono indicate mediante tipi di dato

Qualche esempio:

- Dominio dei numeri naturali e interi
- Dominio dei numeri reali
- …

Sono forniti per convenienza:

- Ogni tipo è associato ad un set di operazioni
- I tipi permettono di effettuare controlli di correttezza
  - Nel caso del C, questi sono svolti durante la compilazione

Tipi di Dato Primitivi in C

Ogni elaboratore fisico:

- È intrinsecamente capace di trattare alcuni tipi di dato
- Questi sono detti **tipi primitivi** dell'elaboratore

I tipi primitivi nel linguaggio C:

- Ricalcano quelli della maggior parte degli elaboratori
- ...Allo scopo di avere la massima efficienza

Tipi di Dato Primitivi in C

In C le categorie di tipi di dato primitivi sono in numero ridotto

Per ogni categoria, vi sono diversi tipi specifici:

- Questi differiscono per il numero di bit utilizzato
  - Su un calcolatore elettronico, ogni informazione
  - …È rappresentata con un numero finito di cifre binarie (0/1)
  - Una cifra binaria si chiama bit

Per ogni tipo specificheremo:

- Il suo nome (in notazione EBNF)
- Il numero di bit
- L'intervallo di valori rappresentabile

Tipi di Dato Primitivi in C

Vediamo i numero interi in C:

Nome	#bit	Intervallo di valori
`short [int]`	≥ 16	Almeno [−32768, +32767]
`int`	≥ 16	Almeno [−32768, +32767]
`long [int]`	≥ 32	Almeno [−2.15e9, +2.15e9]
`long long [int]`	≥ 64	Almeno [-9.22e18, +9.22e18]

- Il numero esatto di bit dipende dal compilatore
  - Tipico: `short: 16, int: 32, long: 64, long long: 64`
- Il tipo `long long` è disponibile solo in C99

Tipi di Dato Primitivi in C

Sono disponibili anche in versione senza segno (naturali)

Nome	#bit	Intervallo di valori
`unsigned short [int]`	≥ 16	Almeno [0, 65535]
`unsigned [int]`	≥ 16	Almeno [0, 65535]
`unsigned long [int]`	≥ 32	Almeno [0, +4.29e9]
`unsigned long long [int]`	≥ 64	Almeno [0, +18.45e18]

- Gli `unsigned` non possono rappresentare numeri negative
- In compenso, posso rappresentare più numeri naturali
- Dimensioni tipiche:
  - `short: 16, int: 32, long: 64, long long: 64`
- `unsigned long long` solo in C99

Tipi di Dato Primitivi in C

Il linguaggio C supporta numeri in virgola mobile ("reali"):

Nome	#bit	Intervallo di valori
`float`	Tipicamente 32	Tipicamente IEEE 754
`double`	Tipicamente 64	Tipicamente IEEE 754
`long double`	80-128	Tipicamente IEEE 754

In realtà, si tratta di numeri razionali, in precisione finita

- Conseguenza: si tratta di una approssimazione!
- Rappresentazione mantissa + esponente
- => Alcuni numeri non sono rappresentabili in modo esatto:
  - 0.1 (periodico – quindi infinito – in forma binaria)
  - π (numero di cifre infinito)

Numeri ed Intevallo Rappresentabile

Con n bit, si possono rappresentare sempre 2^n numeri

Senza entrare troppo nei dettagli:

- Con 2 bit ci sono 4 configurazioni: 00, 01, 10, 11
- Con 3 bit ce ne sono 8: 000, 001, 010, 011, 100, 101, 110, 111

Perché allora ci sono range diversi a parità di numero di bit?

Dipende dalla rappresentazione del numero!

- Non vedremo le rappresentazioni nel dettaglio
- ...Ma faremo alcune considerazioni

Numeri ed Intevallo Rappresentabile

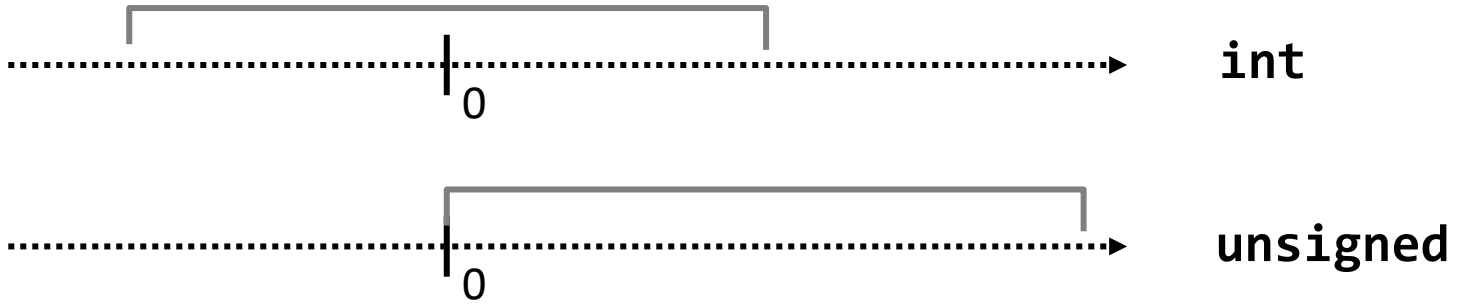
Per gli interi si usa una <u>rappresentazione equispaziata</u>

Per gli interessati:

- Rappresentazione in base 2 per gli unsigned
- Rappresentazione in complemento a 2 per gli int

L'intervallo per gli int e gli unsigned ha la stessa dimensione

Cambia solo la sua collocazione:



Numeri ed Intevallo Rappresentabile

I float/double sono rappresentati con mantissa più esponente

$$\text{mantissa} \times 2^{\text{esponente}}$$

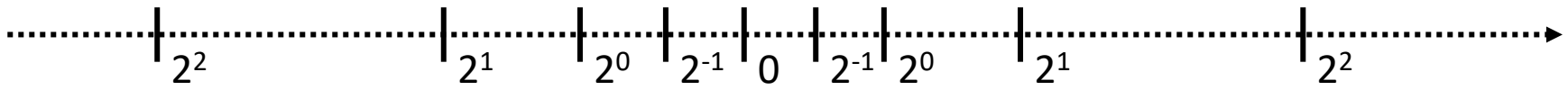
- Alcuni bit si usano per la mantissa altri per l'esponente
- Sia la mantissa che l'esponente sono interi

La rappresentazione del numero in virgola mobile:

- È equispaziata a parità di esponente
- L'esponente decide la distanza tra numeri successivi

Numeri ed Intevallo Rappresentabile

Con n bit, possiamo rappresentare sempre 2^n numeri, ma:



I numeri rappresentabili:

- Sono più fitti vicino allo 0 (esponenti negativi)
- Sono più spaziati lontano dallo 1 (esponenti positivi)

Ne possono seguire errori inaspettati. Alcuni esempi:

- Alcuni numeri non sono rappresentabili
- Sommare numeri molto gradi con molto piccoli può falllire
  - E.g. $2^1 + 2^{-1}$ non è rappresentabile nel nostro esempio!

Tipi di Dato Primitivi in C

Caratteri

Nome	#bit	Intervallo di valori
`char`	Tipicamente 8	Tipicamente [0, 255]

In C, i caratteri sono di fatto numeri interi!

- Sono associati a simboli grafici attraverso una tabella
- Tabella ASCII (pron. «aschi»)
  - ‘A’ ⟷ 65
  - ‘B’ ⟷ 66
  - ‘a’ ⟷ 97
  - ‘b’ ⟷ 98
  - …

Tipi di Dato Primitivi in C

Per curiosità, qui avete l'intera tabella ASCIII

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Tipi di Dato Primitivi in C

Valori Logici (i.e. "vero" e "falso")

Si usano i numeri, con la seguente convenzione:

- 0 corrisponder a "falso"
- Qualunque numero diverso da 0 corrisponde a "vero"
  - Tipicamente si usa 1 o -1

Costanti dei Tipi di Dato Primitivi

Gli interi (`int`) si possono rappresentare in varie basi:

- Decimale: `11, 17, 2345`
- Ottale: `01, 033, 071` (si riconoscono dallo `0` davanti)
- Esadecimale: `0x10, 0xAF, 0xB3` (si riconoscono dallo `0x`)

Si usa la variante più piccola che può contenere il numero

Si possono però specificare alcune eccezioni:

- Long: 10l, 12L (si aggiunge una L dopo il numero)
- Long long: 10ll, 12LL (si aggiungono due L)

Costanti dei Tipi di Dato Primitivi

I numeri in virgola mobile hanno due tipi di rappresentazione

- Notazione decimale: `3.14`, `2.178`
- Notazione scientifica: 0.9e9, 1.33E-4 (i.e. $0.9 \cdot 10^{9}$, $1.33 \cdot 10^{-4}$)

Il default è usare una doppia precisione (double)

Si possono però specificare eccezioni:

- Float: 1.0f, 12.4F (si aggiunge una F dopo il numero)

Costanti dei Tipi di Dato Primitivi

Per i caratteri si usano simboli racchiusi tra apici singoli

- E.g. `'a', 'A', '6'`
- Ricordate che per il C si tratta di numeri
- Ci viene solo fornita un sintassi comda per specificarli

Esistono alcuni caratteri speciali, e.g.:

- `'\n'` → fine linea
- `'\t'` → tabulazione
- `'\''` → apice singolo
- `'\"'` → doppi apici
- `'\\'` → backslash (i.e. \)

Costanti dei Tipi di Dato Primitivi

Stringhe

Una stringa in C è una sequenza di caratteri

- Tra qualche lezione vedremo meglio cosa questo comporti
- A breve le useremo per poter stampare messaggi

Qualche esempio:

- `"ciao"`
- `"mondo\n"` (ciao, quindi a capo)
- `"\tNome\tCognome"` (Nome e Cognome, su due colonne)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Espressioni in C

Espressioni in C

Espressione

Una espressione è una notazione che denota un valore mediante un processo detto valutazione

- Notazione = testo con una sintassi formale
- La valutazione equivale a calcolare l'espressione
- "Denota" significa che produce un risultato (riutilizzabile)

Intuitivamente: sono le espressioni delle scuole medie

- Permettono di svolgere calcoli
- Si basano grossomodo sugli stessi concetti

Espressioni Semplici in C

Ci sono due categorie di espressioni in C:

- Espressioni semplici (o elementari)
- Espressioni composte (combinazione di altre espressioni)

Quali espressioni elementari sono disponibili in C?

Ne abbiamo già visto un esempio: le costanti

```
10
'a'
-0.1
```

Un secondo tipo è dato dalle variabili (che vedremo tra poco)

Chiamate a Funzione in C

- Ci sono due modi di definire espressioni composte in C
- Si possono usare chiamate a funzione, oppure operatori

Una chiamata a funzione ha la sintassi seguente:

```
<nome funzione>(<argomenti>)
<argomenti> ::= <espr.> | <espr.> {, <espr.>}
```

- Le parentesi tonde fanno parte della sintassi
- La chiamata può avere zero o più argomenti (espressioni)
- Se ci sono più argomenti, vanno separati da virgole

Chiamate a Funzione in C

- Ci sono due modi di definire espressioni composte in C
- Si possono usare chiamate a funzione, oppure operatori

Una chiamata a funzione ha la sintassi seguente:

```
<nome funzione>(<argomenti>)
<argomenti> ::= <espr.> | <espr.> {, <espr.>}
```

Qualche esempio:

```
arcsin(0.5)
power(2, 4)
sqrt(9)
```

Chiamate a Funzione in C

Per quanto riguarda la semantica di una chiamata a funzione:

```
<nome funzione>(<argomenti>)
<argomenti> ::= <espr.> | <espr.> {, <espr.>}
```

- Esegue un algoritmo di nome `<nome funzione>`
- Con i parametri di ingresso specificati tra parentesi
- Denota (restituisce) il risultato

I parametri sono altre espressioni!

```
power(sin(0.5), 2)
sqrt(power(3, 3))
```

Operatori in C

Un operatore in C è una funzione con sintassi specializzata

Si usano per le operazioni più comuni. E.g.

```
10 + 2
2 * 3
12 / 4
3 + cos(3.14)
```

Si comportano come le funzioni:

- I parametri si dicono operandi
- Quando vengono valutati denotano (restituiscono) un risultato
- Hanno una implementazione specializzata (molto efficiente)

Classificazione degli Operatori

Gli operatori sono classificati secondo due criteri:

- In base al tipo degli operandi e del risultato
- In base al numero degli operandi

Tipo di operandi/risultato	Numero di operandi
• Aritmetici • Relazionali (di confronto) • Logici • Condizionali • ...	• Unari • Binary • Ternari • ...

Operatori Aritmetici in C

Operandi numerici, risultato numerico

Simbolo	Operazione	Numero di operandi
-	Inversione di segno	unario
+	somma	binario
-	differenza	binario
*	moltiplicazine	binario
/	divisione tra interi	binario
/	divisione in virgola mobile	binario
%	modulo (resto della divisione intera)	binario

- La divisione fra interi è quella delle elementari
  - E.g. 7/3 → 2
- Il modulo è il resto della divisione intera:
  - E.g. 7 % 3 → 1

Operatori Aritmetici in C

Qualche esempio:

- `3 * 4`
- `-(2 * 2)`
- `7+4`
- `10 % 4`

Operatori Aritmetici in C

Qualche esempio:

- `3 * 4` denota 12
- `-(2 * 2)` denota -4
- `7+4` denota 11
- `10 % 4` denota 2

Overloading (Sovraccarico) degli Operatori

In C, lo stesso simbolo può essere associato ad operatori diversi

Un primo esempio:

- Inversione di segno (unario):

```
-5
```

- Differenza (binario)

```
10 - 7
```

Secondo esempi:

- Divisione intera (se gli operandi sono interi)

```
10 / 4 (denota 2)
```

- Divisione in virgola mobile (almeno un operando reale)

```
10.0 / 4.0 (denota 2.5)
16 / 5.0 (denota 3.2)
3F / 4 (denota 0.75)
```

Espressioni Omogenee ed Eterogenee

In C una espressione si dice:

- Omogenea se gli operandi sono tutti dello stesso tipo
- Eterogenea se gli operandi sono di tipo diverso

In caso di espressioni eterogenee:

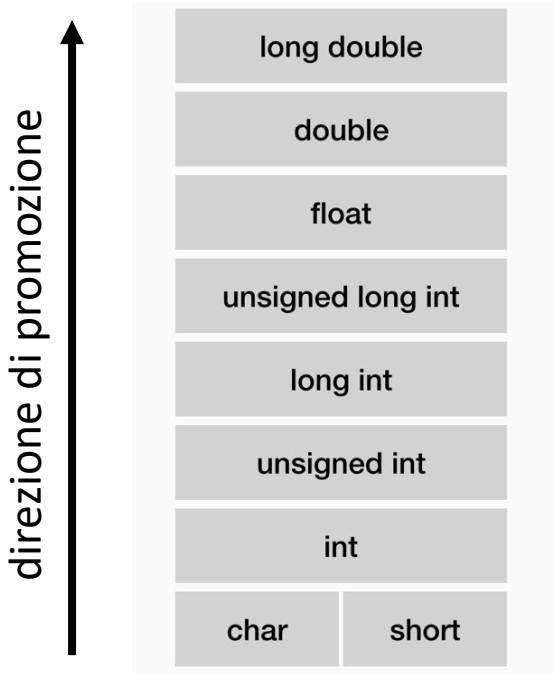
- Il linguaggio tenta una promozione (conversione) di tipo
- Se l'espressione è diventata omogenea, viene valutata
- Altrimenti si tenta una nuova promozione
- Se questo non è fattibile il processo fallisce

Espressioni Omogenee ed Eterogenee

Regole di promozioni di tipo

- Idea: da tipi meno espressivi a tipi più espressivi
- Il risultato è del tipo dell'espressione dopo le promozioni

Nel dettaglio, la catena di promozione è questa:



Espressioni Omogenee ed Eterogenee

Esempio di promozione di tipo:

```
(3 * 2.0) / 5
```

- 3 viene convertito in double (come 2.0)
- Viene valutato il *: il risultato è 6.0 (double)
- 5 viene convertito in double (come 6.0)
- Viene valutato il /: il risultato è 1.2 (double)

Espressioni Omogenee ed Eterogenee

Un secondo esempio di promozione di tipo:

```
(3 / 2) * 5.0
```

- Viene eseguita la divisione intera 3 / 2
- Il risultato è 1 (intero)
- 1 (intero) viene promosso a 1.0 (double)
- 1.0 viene moltiplicato per 5.0
- Il risultato è 5.0

Casting

È possibile forzare una particolare conversione

Basta usare l'operatore di cast, con sintassi:

```
( <tipo> )  <espressione>
```

Esempi:

- `7 / 3` denota 2
- `((float) 7) / 3` denota 2.5

Non si usa spesso, ma in alcuni casi è utile

Operatori Relazionali (di Confronto) in C

Operandi numerici, risultato logico

Simbolo	Operazione	Numero di operandi
==	Uguaglianza	binario
!=	Diversità	binario
>	Maggiore di	binario
<	Minore di	binario
>=	Maggiore o uguale a	binario
<=	Minore o uguale a	binario

I valori logici sono gestiti mediante numeri!

- Denotano 0 se il confronto è falso
- Denotano un valore diverso da 0 se è vero
  - Di solito 1 o -1

Operatori Relazionali (di Confronto) in C

Qualche esempio:

- `1 == 1`
- `4 < 2`
- `3 >= 3`
- `1 != 2`

Operatori Relazionali (di Confronto) in C

Qualche esempio:

- `1 == 1` denota "vero" (i.e. un numero diverso da 0)
- `4 < 2` denota "falso" (i.e. il numero 0)
- `3 >= 3` denota "vero"
- `1 != 2` denota "vero"

Operatori Logici in C

Operandi logici, risultato logico

Simbolo	Operazione	Numero di operandi
!	Negazione	unario
&&	and (e)	binario
||	or (o)	binario

- Di nuovo: i valori logici sono gestiti mediante numeri
- Sono utili per controllare condizioni complesse:

```
(2 < 3) || (3 != 0)
(2 <= 3) && (3 <= 4)
```

Operatori Logici in C

Qualche esempio:

- `(1 <= 2) && (2 <= 3)`
- `(1 <= 2) && (3 >= 2)`
- `(1 <= 2) || (3 >= 2)`
- `!(1 == 1)`

Operatori Logici in C

Qualche esempio:

- `(1 <= 2) && (2 <= 3)` denota "vero"
- `(1 <= 2) && (3 >= 2)` denota "falso
- `(1 <= 2) || (3 >= 2)` denota "vero"
- `!(1 == 1)` denota "falso"

Occhio a Questi Tre

Quattro operatori a cui fare attenzione:

Operatore di uguaglianza: ==

- Non si usa =, perché è riservato per l'assegnamento
- Ne parleremo tra poco

Operatori and e or: && e ||

- Esistono anche & e | (simbolo ripetuto una solta volta)
- ...Ma fanno tutt'altro (non ne parleremo)

Operatore ^:

- Non è l'elevamento a potenza, ma tutt'altro!

Priorità degli Operatori

La 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 avere egual priorità

Associatività degli Operatori

L'associatività specifica l'ordine di valutazione degli operatori
...quando compaiono operatori (infissi) di egual priorità

- Un operatore può essere associativo a sinistra o a destra

Esempio:

```
3 - 10 + 8
```

- Si legge come (3 - 10) + 8
- ...Perché gli operatori - e + sono equiprioritari e associativi a sinistra

Priorità, Associatività e Parentesi

- Priorità ed associatività servono per interpretare le espressioni
- ...Ma possono essere alterate mediante l'uso delle parentesi

Esempio:

```
(3 + 10) * 20
```

- Denota 260 (anziché 203)

Esempio:

```
30 - (10 + 8)
```

- Denota 12 (anziché 28)

Priorità ed Associatività degli Operatori in C

- In generale valgono le regole della matematica

Priorità	Simbolo	Operazione	Associatività
1	( )	Chiamata a funzione	sx
1	.	Indicizzazione	sx
1	.	Selezione (strutture)	sx
1	->	Selezione (puntatori a struttura)	sx
2	!	Negazione	dx
2	+	Mantenimento di segno	dx
2	-	Inversione di segno	dx
2	++	Incremento (postfisso e prefisso)	dx
2	--	Decremento (postfisso e prefisso)	dx
2	*	Indirizzamento	dx
2	&	Dereferenziamento	dx
2	sizeof	Dimensione	dx

operatori unari

Priorità ed Associatività degli Operatori in C

moltiplicatvi

Priorità	Simbolo	Operazione	Associatività
3	*	Moltiplicazione	sx
3	/	Divisione (intera)	sx
3	/	Divisione (virgola mobile)	sx
3	%	Modulo	sx
4	+	Somma	sx
4	-	Sottrazione	sx
5	<<	Shift sx	sx
5	>>	Shift dx	sx
6	<	Minore	sx
6	>	Maggiore	sx
6	<=	Minore o uguale	sx
6	>=	Maggiore o uguale	sx

Priorità ed Associatività degli Operatori in C

Priorità	Simbolo	Operazione	Associatività
7	==	Uguaglianza	sx
7	!=	Diverso	sx
8	&	AND bit a bit	sx
9	^	XOR bit a bit	sx
10	|	OR bit a bit	sx
11	&&	AND logico	sx
12	||	OR logico	sx
13	? :	Condizionale	dx
14	=	Assegnamento (incl. +=, -=, etc.)	dx
15	,	Concatenazione	sx

- La lista è abbastanza complete
- Include operatori che non useremo nel corso

Espressioni & Istruzioni

Una espressione seguita da ";" è una istruzione

- Quando viene eseguita viene valutata
- …E non fa altro (non è particolarmente utile)

Qualche esempio (sano):

```
int main() {
    3 * 2 – 1;
    7 / 2 * 2;
    7 / 2.0 * 2;
    7 / 2 * 2.0;
    13 % 2 == 1;
    1 <= 3 && 2 <= 4;
}
```

Espressioni & Istruzioni

Una espressione seguita da ";" è una istruzione

- Quando viene eseguita viene valutata
- …E non fa altro (non è particolarmente utile)

Qualche esempio (sano):

```
int main() {
    3 * 2 – 1;          // 6 – 1 → 5
    7 / 2 * 2;          // 3 * 2 → 6
    7 / 2.0 * 2;        // 3.5 * 2 → 7.0 (double)
    7 / 2 * 2.0;        // 3 * 2.0 → 6.0 (double)
    13 % 2 == 1;        // 1 == 1→ vero
    1 <= 3 && 2 <= 4;   // vero && vero → vero
}
```

Espressioni & Istruzioni

Una espressione seguita da ";" è una istruzione

- Quando viene eseguita viene valutata
- ...E non fa altro (non è particolarmente utile)

Qualche esempio (malsano):

```
int main() {
    2 <= 3 <= 4;
    (2 != 3) * 4;
    1 < 3 == 2 < 4;
}
```

Espressioni & Istruzioni

Una espressione seguita da ";" è una istruzione

- Quando viene eseguita viene valutata
- …E non fa altro (non è particolarmente utile)

Qualche esempio (malsano):

```
int main() {
    2 <= 3 <= 4;          // (2 <= 3) <= 4 (brutta sorpresa)
                          // vero <= 4 (mal definito!)
    (2 != 3) * 4;         // vero * 4 (mal definito!)
    1 < 3 == 2 < 4;       // vero == vero (ok, ma al limite)
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

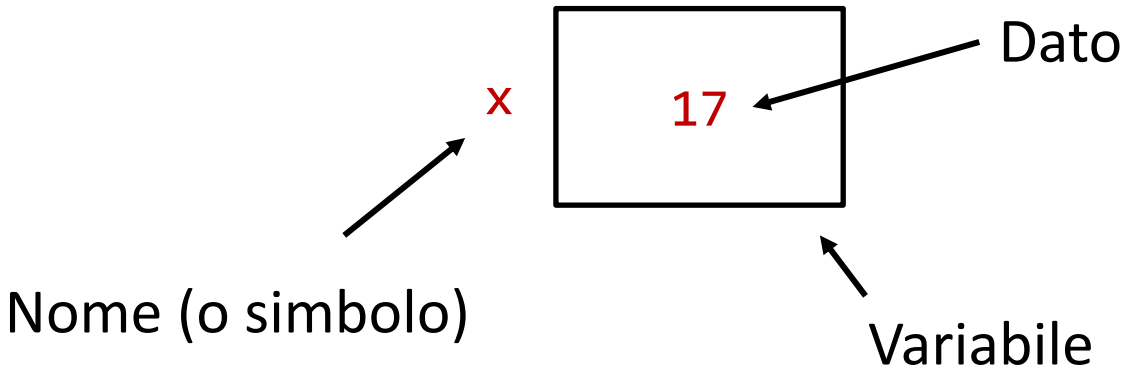
Variabili in C

Cosa Fare con i Dati?

Per implementare algoritmi è essenziale poter memorizzare dati

In C, è possibile farlo attraverso un costrutto detto variabile

- In matematica, una variabile è un simbolo associato ad un dato
- In C, una variabile è un contenitore con nome per un dato



Cosa Fare con i Dati?

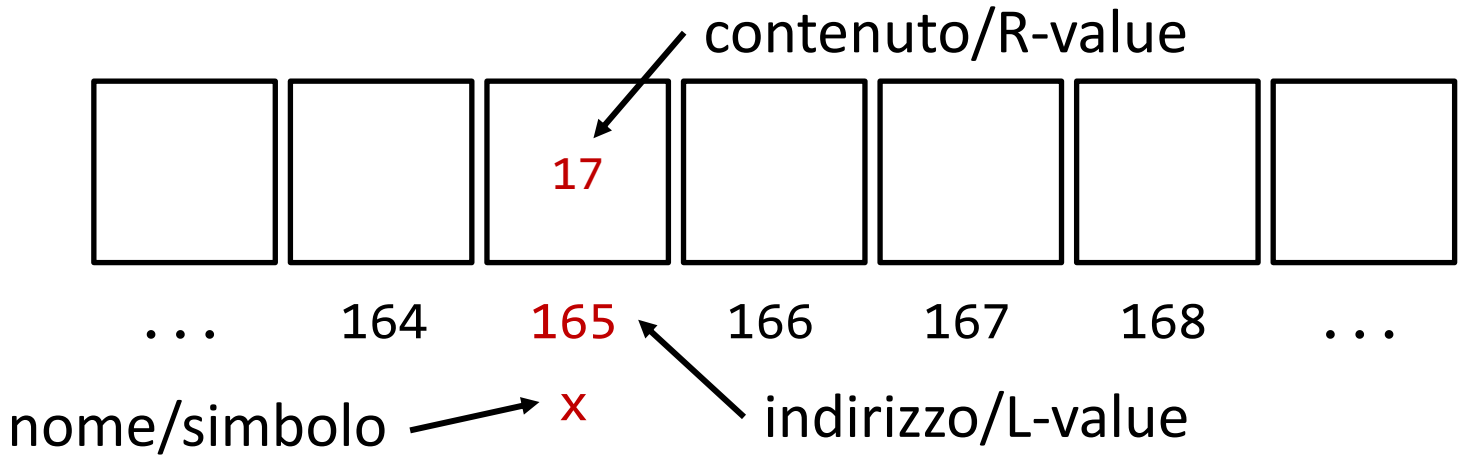
Una variabile è un <u>contenitore:</u>

- ...Quindi il suo contenuto può essere cambiato!
- Una variabile può valere prima 17, poi 4, poi 0, etc.
- In matematica non è così

Variabili in C, più precisamente

Una variabile è una astrazione di una (o più) celle di memoria

- La memoria è vista come una collezione di celle contigue
- Ogni cella ha un indirizzo
- Una variabile associa un nome/simbolo ad un indirizzo
  - L'indirizzo si dice anche L-value della variabile
  - Il contenuto della cella si dice anche R-value della variabile

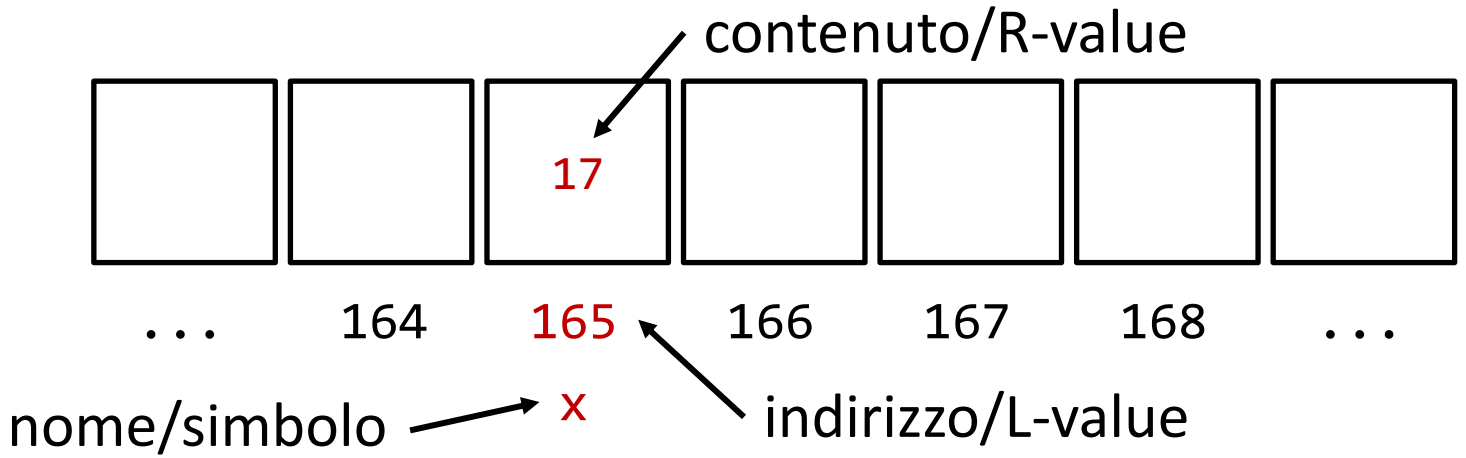


Variabili in C, più precisamente

Una variabile è una astrazione di una (o più) celle di memoria

L'associazione tra nome ed indirizzo:

- È determinata a tempo di compilazione
- È immutabile per il tempo di vita del programma

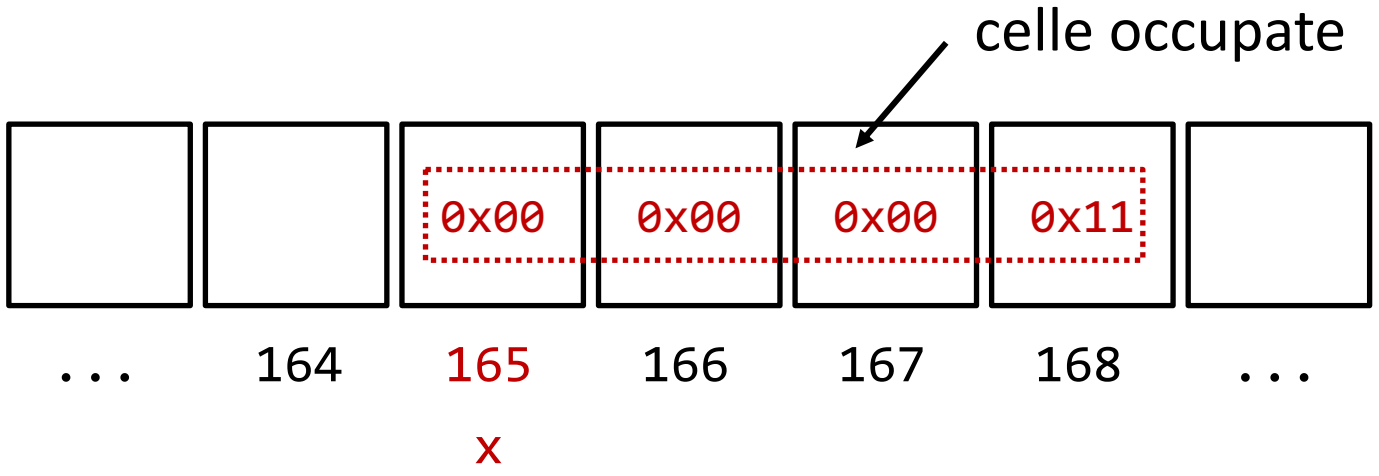


Variabili in C, più precisamente

Una variabile è una astrazione di una (o più) celle di memoria

Una variabile può in realtà occupare più celle

- Ogni cella è un byte (8 bit)
- Il numero di celle occupate dipende dal tipo del dato
  - E.g. intero 32 bit = 4 celle
- Ad essere memorizzata è la rappresentazione binaria del dato



Dichiarazione/Definizione di Variabili

Per utilizzare una variabile, bisogna prima costruirla

- In C lo si può fare con una istruzione specifica:
- Questa è detta "dichiarazione" o "definizione"

Sintassi di una dichiarazione/definizione di variabile:

```
<tipo> <nome>;
```

Qualche esempio:

```
int x;
float y;
char c;
```

Dichiarazione/Definizione di Variabili

Semantica di una dichiarazione/definizione:

Quando una definizione nella forma seguente viene eseguita:

```
<tipo> <nome>;
```

- Viene predisposto spazio in memoria per un dato
- ...di dimensione pari a quella di `<tipo>`
- Quindi l'indirizzo della prima cella viene associato a `<nome>`

Il C è un linguaggio fortemente tipizzato:

- Ogni variabile ha un tipo e questo è immutabile
- Ciò permette controlli accurati in fase di compilazione

Variabili ed Identificatori

Tecnicamente, il nome di una variabile è un identificatore

Tutti gli identificatori in C condividono la stessa sintassi:

```
<identificatore> ::=
      <lettera> | _ {<lettera> | _ | <cifra> }
```

Qualche esempio:

- `somma` (valido)
- `c1p8` (valido)
- `_resto` (valido)
- `1var` (non valido)
- `_1234` (valido, ma malvagio)

Variabili ed Identificatori

Quando avete tempo, leggete questo:

"Come scrivere programmi incomprensibili" (di Davide Bianchi)

http://www-lia.deis.unibo.it/Courses/RetiLM/Come%20scrivere%20programmi%20incomprensibili.htm

Un assaggio:

> ***Nuovi usi per il "libro dei nomi" (ISBN: 0-380-78047-X)***
>
> *Comperatevene subito una copia e non vi troverete mai in difficolta' nell'inventare il nome di una variabile. "Gino" e' un ottimo nome, e' breve, si scrive alla svelta ed ha una marea di altre possibilita' (Lino, Pino, Dino...) tutte molto simili e facilmente confondibili tra di loro. Se state cercando per dei nomi "facili" provate "adsf" o "aeou"*

Variabili in Espressioni

Il nome di una variabile può comparire in una espressione:

```
0.5 * h * l
sin(2 * pi)
```

- Una variabile può essere usata come espressione semplice
- Denota il valore contenuto al momento della valutazione
  - E.g. 3.14189 per `pi`...
  - ...Ma solo se pi contiene effettivamente quel numero!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Operatore di Assegnamento

Operatore di Assegnamento

L'operatore di assegnamento (i.e. =):

- Serve a cambiare il contenuto della variabile
- Ha la sintassi:

```
<variabile> = <espressione>
```

Quando l'operatore viene eseguito:

- Valuta `<espressione>`
- Inserisce il valore denotato in `<variabile>`
- Denota il valore (i.e. lo restituisce)

Operatore di Assegnamento

Vediamo qualche esempio:

```
a = 10;
```

- Nella variabile "a" viene inserito il valore 10

```
a = 10;
a = -2;
```

- Dopo il primo assegnamento "a" contiene "10"
- Dopo il secondo assegnamento "a" contiene -2

R-value e L-value

Rispetto all'operatore di assegnamento

Una varabile può comparire a destra o a sinistra

```
a = a + 1;
```

- Quando compare a destra, è una espressione semplice
  - Denota (come già detto) il valore contenuto
  - Non a caso lo abbiamo chiamato R-value!
- Quando compare a sinistra, indica dove scrivere
  - Rappresenta l'indirizzo della cella di memoria
  - Non a caso lo abbiamo chiamato L-value!

R-value e L-value

Un operatore di assegnamento:

- Valuta sempre l'espressione alla sua destra
- Scrive il valore denotato nella variabile specifica a sinistra

Es. 1: L'operatore non rappresenta una uguaglianza matematica:

```
a = b
```

- Scrive il contenuto di "b" in "a"
- Non indica che "a" e "b" sono uguali

Es. 2: questa notazione non ha senso:

```
2 = b
```

- "b" è una espressione, ma "2" non è una variabile

Inizializzazione di una variabile

Si può assegnare un valore ad una variabile alla sua definizione

Si dice inizializzazione. Un esempio:

```
int a = 10;
```

Definisce "a" e vi inserisce un valore

Una variabile non inizializzata non è vuota:

- Ha il contenuto presente in memoria nella cella allocata
- Tale contenuto non è controllabile (lo si può pensare casuale)
- Per non avere sorprese, inizializzate le variabili

Assegnamenti come Espressioni

In C l'operatore di assegnamento è una espressione

- Denota il valore inserito nella variabile

Può comparire in espressioni:

```
3 * (a = 2)
```

- Denota 3 * 2 (e inserisce 2 in "a")

È associativo a destra:

```
a = b = 2
```

- Corrisponde a "a = (b = 2)", inserisce 2 in "b", quindi in "a"

Evitate di usarlo in questo modo (poco leggibile)

Espressioni con Effetti Collaterali

L'assegnamento è una espressione con effetti collaterali

- Quando viene valutata, oltre a denotare un valore
- ...Altera il contenuto di una variabile

In C vi sono altre espressioni del genere:

- Operatore di incremento ++
- Operatore di decremento –
- Operatori di assegnamento compatti

Incremento e Decremento

Gli operatori di incremento e decremento hanno due varianti:

Variante postfissa:

`i++` oppure `i--`

- Incrementa/decrementa "i"
- Denota il valore di "i" prima all'incremento

Variante prefissa:

`++i` oppure `--i`

- Incrementa/decrementa "i"
- Denota il valore di "i" dopo all'incremento

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```

i vale 5, k vale 6

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```

i vale 5, k vale 6

```
int i=4, j, k = 5;
j = i + k++;
```

j vale 9, k vale 6

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```

i vale 5, k vale 6

```
int i=4, j, k = 5;
j = i + k++;
```

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```

i vale 5, k vale 6

```
int i=4, j, k = 5;
j = i + k++;
```

j vale 9, k vale 6

```
int j, k = 5;
j = ++k - k++;
```

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```
i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```
i vale 5, k vale 6

```
int i=4, j, k = 5;
j = i + k++;
```
j vale 9, k vale 6

```
int j, k = 5;
j = ++k - k++;
```
Dipende dal compilatore (EVITARE!)

Operatori di Assegnamento Compatti

Una notazione compatta per operatori binari + assegnamento:

```
<variabile> <op>= <espressione>
```

Viene espansa in :

```
<variabile> = <variabile> <op> <espressione>
```

Esempi:

```
k += j     →    k = k + j
k *= (a+b) →    k = k * (a + b)
```

Compatibilità di Tipo

E se il tipo dell'espressione e quello della variabile sono diversi?

Ci sono due casi possibili:

- Il tipo della variabile è più espressivo dell'espressione

```
float a = 5 * 2;
```

  - Viene effettuata una promozione di tipo
  - Nessun problema particolare

- Il tipo della variabile è meno espressivo dell'espressione

```
int a = 5 / 2.0;
```

  - Si perde informazione!
  - In questo caso, "a" conterrà 2 invece che 2.5 (troncamento)

Un Semplice Esempio

- Data una temperatura espressa in gradi Celsius
- ...calcolare il corrispondente valore in gradi Fahrenheit

Soluzione:

```
int main() {
        float c = 18;

        /* Vale l'uguaglianza: c * 9/5 = f – 32 */

        float f = 32 + c * 9/ 5.0;
}
```

Un Altro Esempio

```
int X,Y;
unsigned int Z;
float SUM;

X=27;
Y=343;
Z = X + Y -300;
X = Z / 10 + 23;
Y = (X + Z) / 10 * 10;
```

Un Altro Esempio

```
int X,Y;
unsigned int Z;
float SUM;

X=27;
Y=343;
Z = X + Y -300;
X = Z / 10 + 23;
Y = (X + Z) / 10 * 10;        qui X=30, Y=100, Z=70
X = X + 70;
Y = Y % 10;
Z = Z + X -70;
```

Un Altro Esempio

```
int X,Y;
unsigned int Z;
float SUM;

X=27;
Y=343;
Z = X + Y -300;
X = Z / 10 + 23;
Y = (X + Z) / 10 * 10;        qui X=30, Y=100, Z=70
X = X + 70;
Y = Y % 10;
Z = Z + X -70;                Z vale 100
SUM = Z * 10;                 qui X=100, Y=0, Z=100 , SUM =1000.0
```

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono	Neutrale Buono	Caotico Buono
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono	Caotico Buono
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio `i = i + i/i`	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio `i = i + i/i`	Neutrale Malvagio `i += 73*139%2`	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio `i = i + i/i`	Neutrale Malvagio `i += 73*139%2`	Caotico Malvagio `i = ++i--`

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>