



- 2° Modulo – parte 2^
  - La rappresentazione dell'Informazione
  - Struttura dei programmi C
  - Tipi Semplici del C
  - Costanti Variabili, Operatori
  - Espressioni
  - Operatori aritmetici, logici, bitwise

<http://www.elet.polimi.it/upload/martucci/index.html>

## STRUTTURA DI UN PROGRAMMA C - 2

### Parte dichiarativa globale:

- servizi (funzioni) importate da altri moduli (file), cioè definite e codificate in altri file
- «oggetti» (tipi di dati, variabili, costanti simboliche, prototipi di funzioni) visibili (utilizzabili) da tutto il programma, cioè da main e dalle altre funzioni.

### Programma principale:

```
main ()
```

```
{
```

```
  parte dichiarativa  
  locale
```

```
  parte esecutiva
```

```
}
```

parola riservata (identificatore di funzione)  
appare una e una sola volta nel programma  
definisce l'inizio dell'esecuzione  
è (formalmente) una funzione

definisce l'insieme di «oggetti» usati dal  
programma principale per l'esecuzione.  
sono oggetti visibili (locali) a main.

insieme di istruzioni che costituiscono il  
programma principale



# Parte dichiarativa – I DATI

---

- **Costanti**

- Non si dichiarano, autodefinite nel contesto

- **Variabili**

- Si dichiarano con NOME e TIPO
- Tipi Semplici

## Commenti:

Sono sequenze di caratteri ignorate dal compilatore.  
Vanno racchiuse tra `/* ... */`:

```
/* questo e`  
   un commento  
dell'autore */
```

I commenti vengono generalmente usati per introdurre note esplicative nel codice di un programma.

## Costanti

### Numeri interi

Rappresentano numeri relativi (quindi con segno):

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

### Numeri reali

Varie notazioni:

24.0      2.4E1      240.0E-1

**Suffissi:** l, L, u, U (interi-long, unsigned)  
f, F (reali - floating)

**Prefissi:** 0 (ottale)      0x, 0X(esadecimale)

## Caratteri:

Insieme dei caratteri disponibili (e` dipendente dalla implementazione). In genere, ASCII esteso (256 caratteri). Si indicano tra singoli apici:

'a' 'A'

## Caratteri speciali:

newline	\n
tab	\t
backspace	\b
form feed	\f
carriage return	\r
codifica ottale	\ooo (o cifra ottale 0-7)
	\041 è la codifica del carattere !

Il carattere \ inibisce il significato predefinito di alcuni caratteri "speciali" (es. ', ", \, ecc.)

' \ \ " \0 (carattere nullo)

## Stringhe:

Sono sequenze di caratteri tra doppi apici " ".

"a" "aaa" "" (stringa nulla)

## Esempio: (printf e` l'istruzione per la stampa)

```
printf("Prima riga\nSeconda riga\n");  
printf("\\\\"/");
```

## Effetto ottenuto:

```
Prima riga  
Seconda riga  
\"/
```

# Constant Values

---

- Integral bases: 68, 043, 0x3B
- Integral suffixes: -45L, 88u, 4u1
- Floating point suffixes (default is double): 45.f, .34L, 0.87E-2
- Characters: 'a', '\n', '\"'
- Strings (null-terminated): "", "I am a string", "\n",  
"tab here:\t"

Constants values are (obviously) non-addressable.

# Constants

## Integer vs Floating-point Constants

10 33 3.333

## Decimal Constants

10 33L

## Octal

017

## Hex

0X1A 0x1A 0x1a

## Character

'A' 'a' '2'

## String

"A string has more than one character"



# Parte dichiarativa – I DATI

---

- Costanti
  - Non si dichiarano, autodefinite nel contesto
- Variabili
  - Si dichiarano con NOME e TIPO
  - Tipi Semplici

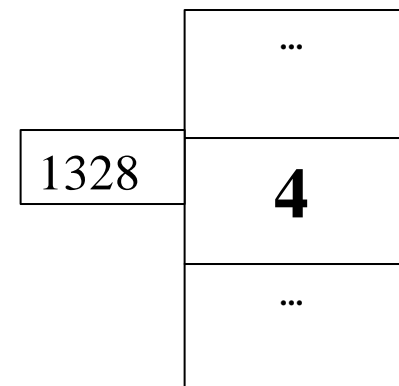


# Variabili

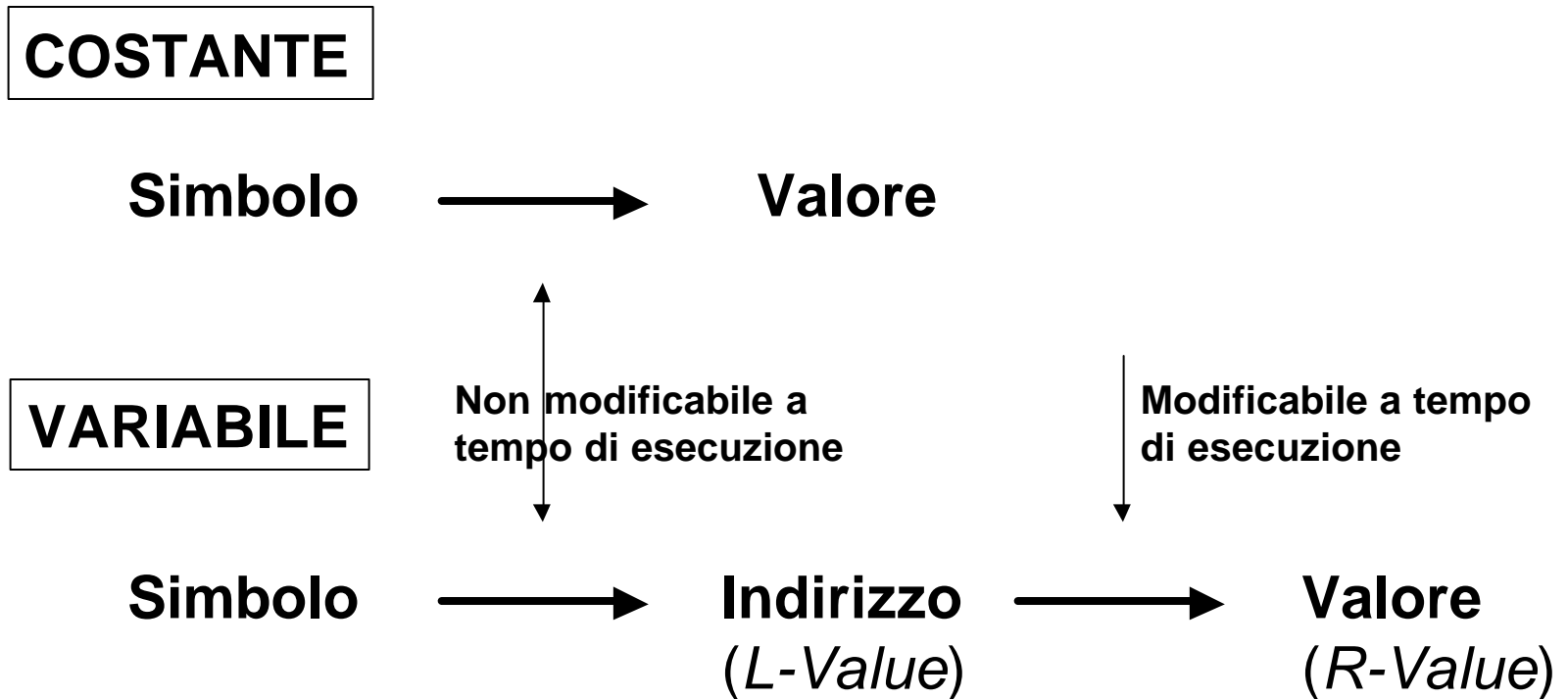
- Una *variabile* è un'astrazione della cella di memoria
- Formalmente, una **variabile** è un **simbolo** associato a un *indirizzo fisico* (**L-VALUE**)

<i>simbolo</i>	<i>indirizzo</i>
<b>x</b>	<b>1328</b>

- che **denota un valore** (**R-VALUE**)
- perciò, l' R-VALUE di x è attualmente 4:



# Costanti e Variabili



# Costanti e Variabili

- Una **costante** è un'astrazione simbolica di un valore
- L'associazione simbolo-valore non cambia mai durante l'esecuzione
- Una **variabile** è un **simbolo** associato a un *indirizzo fisico* (**L-VALUE**) che contiene un valore (**R-VALUE**)
- L'associazione **simbolo-indirizzo** non cambia mai durante l'esecuzione, ma può cambiare l'associazione **indirizzo-valore**
- Pertanto, nel caso di variabile, ad uno stesso simbolo possono corrispondere valori differenti in diversi momenti dell'esecuzione del programma

## Attenzione

- **R-VALUE** può cambiare nel corso dell'esecuzione
- **L-VALUE** è fissato (e non cambia durante l'esecuzione)

# Dichiarazione di variabili

## Scopo:

- Elencare tutte le *variabili* che saranno utilizzate nella parte esecutiva
- Attribuire ad ogni variabile un **tipo**
- E' possibile raggruppare le dichiarazioni di più variabili dello stesso tipo in una lista separata da ,

## Esempio

```
int a;
```

```
float x, y;
```

```
char c, w, z;
```

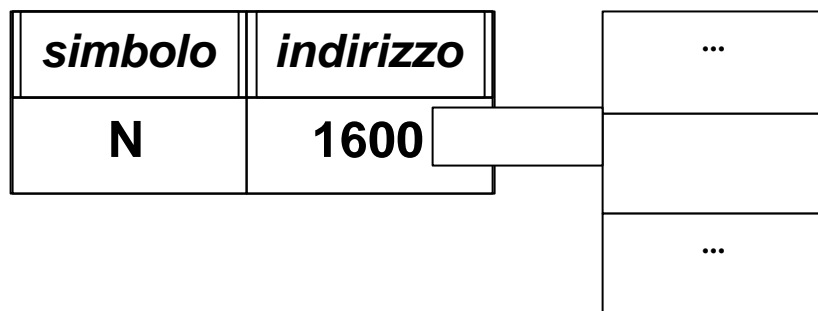
Cercare di utilizzare sempre nomi significativi per le variabili!

# Istruzione di assegnamento

- Denotata mediante il simbolo =  
(l'operatore relazionale di uguaglianza è denotata con il simbolo == )
- Viene utilizzata per assegnare ad una variabile (non ad una costante!) il valore di un'espressione

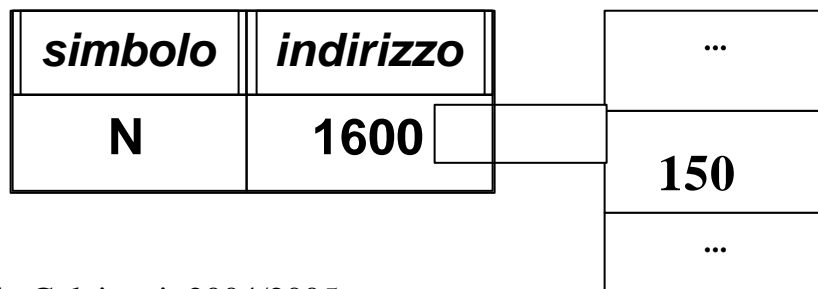
## Esempio

`int N;`



L'esecuzione di una **dichiarazione** provoca l'allocazione di uno spazio in memoria equivalente a quello necessario a contenere un dato del tipo specificato

`N = 150;`



L'esecuzione di un **assegnamento** provoca l'inserimento nello spazio relativo alla variabile del valore indicato a destra del simbolo =

## Istruzione di assegnamento (*cont.*)

- L'esecuzione di un'istruzione di assegnamento comporta innanzitutto la valutazione di tutta l'espressione a destra dell'assegnamento.

Es.,

$$c = 2;$$
$$d = (c+5)/3 - c;$$
$$d = (d+c)/2;$$

- Dopodiché, si inserisce il valore risultante nella locazione di memoria relativa alla variabile (posta a sinistra dell'assegnamento)
- Il primo assegnamento di un valore ad una variabile dichiarata viene detto **inizializzazione**.

**In C, l'inizializzazione si può effettuare anche al momento della dichiarazione.**

Es.,

```
int a, b=56;
```

## Istruzione di Assegnamento

Il concetto di **variabile** nel linguaggio C rappresenta un'astrazione della cella di memoria.

L'istruzione di **assegnamento**, quindi, e' l'astrazione dell'operazione di scrittura nella cella che la variabile rappresenta.

### Assegnamento:

```
<identificatore-variabile> = <espressione>
```

### Esempi:

```
main()
{
  int a; /* definizione di a */
  ...
  a=100; /*assegnamento ad a del
         valore 100 */
}
```

```
#include <stdio.h>
main()
{
  float X, Y;

  /* assegnamento del risultato di una
  espr. aritmetica: */

  Y = 2*3.14*X;
}
```

## Costanti

Una **costante** rappresenta un dato che **non** puo' cambiare di valore nel corso dell'esecuzione.

La dichiarazione di una costante associa ad un identificatore (**nome** della costante) un **valore** (espresso eventualmente mediante altra costante).

```
<dich-costante> ::=
const <tipo> <identificatore-costante>=<costante>
<costante> ::= (+ | -) <identificatore-costante> |
(+|-)<numero-senza-segno> | <altre-costanti>
```

```
const float pigreco = 3.14;
```

```
const float pigreco = 3.1415926; e = 2.7182;
menoe = - e;
```

☞ Anche in questo caso, prima di essere usato, un identificatore deve essere gia' stato definito (ad es., e per definire menoe).

### Vantaggi derivanti dall'uso di costanti:

Leggibilita' e modificabilita' dei programmi.

## Esempio:

```
#include <stdio.h>
main()
{
/* programma che, letto un numero a
terminale stampa il valore della
circonferenza del cerchio con quel raggio
*/

const float pigreco = 3.1415926;
float X, Y;

scanf("%f",&X); /*legge X */
Y = 2*pigreco*X;
printf("%f",Y); /* stampa Y */
}
```

## Tipo di dato

Un **tipo di dato** T e` definito come:

- Un insieme di valori D (**dominio**)
- Un insieme di funzioni (**operazioni**)  $f_1, \dots, f_n$ , definite sul dominio D;

### In pratica:

Un tipo T e` definito:

- dall'**insieme di valori** che le variabili di tipo T possono assumere;
- dall'**insieme di operazioni** che possono essere applicate ad operandi del tipo T.

### Esempio:

Consideriamo i numeri *naturali*

**Tipo\_naturali** = [N, {+, -, \*, /, =, >, <, etc }]

- N e` il dominio
- {+, -, \*, /, =, >, <, etc } e` l'insieme di operazioni



## Il concetto di Tipo

Un linguaggio di programmazione è *tipato* se prevede costrutti specifici per attribuire tipi ai dati utilizzati nei programmi.

### Se un linguaggio è tipato:

- ☞ Ogni dato (variabile o costante) del programma deve appartenere ad **uno ed un solo** tipo.
- ☞ Ogni operatore richiede **operandi** di tipo specifico e produce **risultati** di tipo specifico.

### Vantaggi:

- ☞ **Astrazione:** L'utente esprime e manipola i dati ad un livello di astrazione più alto della loro organizzazione fisica. Maggior portabilità.
- ☞ **Protezione:** Il linguaggio protegge l'utente da combinazioni errate di dati ed operatori (**controllo statico** sull'uso di variabili, etc. in fase di compilazione).
- ☞ **Portabilità:** l'indipendenza dall'architettura rende possibile la compilazione dello stesso programma su macchine profondamente diverse.

## Tipo di Dato in C

Il C è un linguaggio tipato.

### Classificazione dei tipi di dato in C:

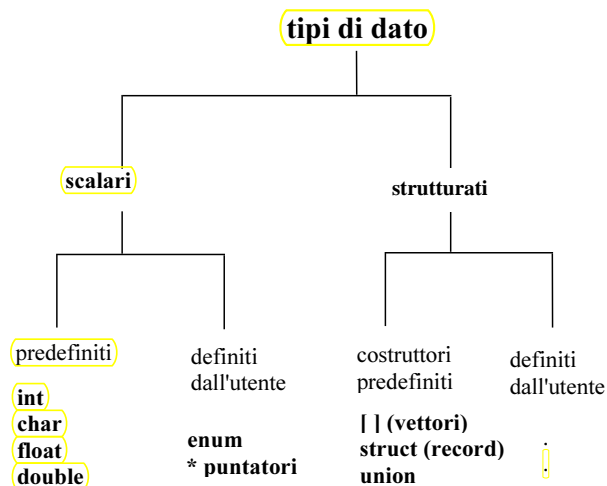
#### Si distingue tra:

- tipi **primitivi** sono tipi di dato previsti dal linguaggio (built-in) e quindi rappresentabili direttamente.
- tipi **non primitivi** sono tipi *definibili dall'utente* (mediante appositi costruttori di tipo, v. *typedef*).

#### Inoltre, si distingue tra:

- tipi **scalari**, il cui dominio è costituito da elementi *atomici*, cioè logicamente non scomponibili.
- tipi **strutturati**, il cui dominio è costituito da elementi non atomici (e quindi scomponibili in altri componenti).

## Classificazione dei tipi di dato in C



## Tipi primitivi

Il C prevede quattro tipi primitivi:

- **char** (caratteri)
- **int** (interi)
- **float** (reali)
- **double** (reali in doppia precisione)

È possibile applicare ai tipi primitivi dei **quantificatori** e dei **qualificatori**:

### Quantificatori:

- I **quantificatori** (*long* e *short*) influiscono sullo spazio in memoria richiesto per l'allocazione del dato.
  - **short** (applicabile al tipo **int**)
  - **long** (applicabile ai tipi **int** e **double**)

### Esempio:

```
int X;      /* se X e' su 16 bit..*/
long int Y; /* ..Y e' su 32 bit */
```

### Qualificatori:

- I **qualificatori** condizionano il dominio dei dati:
  - **signed** (applicabile ai tipo **int** e **char**)
  - **unsigned** (applicabile ai tipo **int** e **char**)

```
int A;      /*A in[-2e15,2e15-1] */
unsigned int B; /*B in[0,2e16-1]*/
```

## TIPI SEMPLICI BUILT-IN DEL C

I nomi di questi tipi sono delle parole chiave del linguaggio:

- **char**: (8 bit - 1 byte) Valori da 0 a 255 che rappresentano la codifica ASCII estesa del carattere corrispondente
- **int** (16bit - 2 byte) Rappresentano gli interi relativi. Valori in complemento a 2 da - 32768 a + 32767
- **float** (32 bit - 4 byte). Rappresentano i razionali espressi in virgola mobile (buona approssimazione dei reali). Valori espressi tramite mantissa e esponente (standard IEEE). Intervallo di valori rappresentabili: da  $-10^{38}$  a  $+ 10^{38}$
- **double** (8 byte). Sono float in doppia precisione.

Si dicono tipi aritmetici (char, int *integral*; float e double *floating*)

L'insieme di valori ammissibili (vmin e vmax) e lo spazio allocato in memoria possono essere modificati tramite **qualificatori** (specificatori) di tipo. I qualificatori sono parole chiave del linguaggio che si premettono al tipo.

### Indirizzo di una variabile

- operatore: `&nome_var`
- valori assunti per gli indirizzi: interi  $\geq 0$

`&nome_var` rappresenta l'indirizzo di memoria del primo byte allocato per la variabile.



## **Data Type Qualifiers**

short

long

signed

unsigned

short int <= int <= long int



# Sintassi generica

Sintassi generica per dichiarare una variabile:

```
{classe} [tipo] [nome] {=valore};
```

- ✓ {...} indica parametro opzionale
- ✓ [...] indica parametro obbligatorio



# Esempi

```
(long) int i=1e50; //aumento il valore massimo
```

```
(short) int i=10; // diminuisco il valore massimo
```

```
(unsigned) char carattere='a';
```

```
(const) double val=123.42134;
```

```
(long) double val =1234.42;
```

**!!** (Alcuni modificatori non hanno senso:

```
(long char) a; /* non ha senso */
```

## Il tipo `int`

### Dominio:

Il dominio associato al tipo `int` rappresenta l'insieme dei numeri interi (cioè  $\mathbb{Z}$ , insieme dei numeri relativi): ogni variabile di tipo `int` è quindi l'astrazione di un intero.

**Esempio:** definizione di una variabile intera

```
int A; /* A e' un dato intero */
```

☞ Poiché si ha sempre a disposizione un numero **finito** di bit per la rappresentazione dei numeri interi, il dominio rappresentabile è di estensione finita.

### Ad esempio:

se il numero  $n$  di bit a disposizione per la rappresentazione di un intero è 16, allora il dominio rappresentabile è composto di:

$$(2^n - 1) = 2^{16} - 1 = 65.536 \text{ valori}$$

### Uso dei quantificatori `short/long`:

Aumentano/diminuiscono il numero di bit a disposizione per la rappresentazione di un intero:

$\text{spazio}(\text{short int}) \leq \text{spazio}(\text{int}) \leq \text{spazio}(\text{long int})$

### Uso dei qualificatori:

- **signed:** viene usato un bit per rappresentare il segno. Quindi l'intervallo rappresentabile è:

$$[-2^{n-1}-1, +2^{n-1}-1]$$

- **unsigned:** vengono rappresentati valori a priori positivi. Intervallo rappresentabile:

$$[0, (2^n - 1)]$$

## Il tipo int

### Operatori:

Al tipo **int** (e tipi ottenuti da questo mediante qualificazione/quantificazione) sono applicabili i seguenti operatori:

### Operatori aritmetici:

forniscono risultato intero:

$+$ ,  $-$ ,  $*$ ,  $/$       somma, sottrazione, prodotto, divisione intera.

$\%$       operatore *modulo*: resto della divisione intera:

$10\%3 \rightsquigarrow 1$

$++$ ,  $--$       *incremento* e *decremento*: richiedono un solo operando (una variabile) e possono essere postfissi ( $a++$ ) o prefissi ( $++a$ ) (v. espressioni)

### Operatori relazionali:

si applicano ad operandi interi e producono risultati “*booleani*” (cioè, il cui valore può assumere soltanto uno dei due valori {*vero*, *falso*}):

$==$ ,  $!=$       uguaglianza, disuguaglianza:

$10==3 \rightsquigarrow falso$

$10!=3 \rightsquigarrow vero$

$<$ ,  $>$ ,  $<=$ ,  $>=$       minore, maggiore, minore o uguale, maggiore o uguale

$10>=3 \rightsquigarrow vero$



## I tipi float e double (reali)

### Dominio:

Concettualmente, e' l'insieme dei numeri reali R.

In realta', e' un sottoinsieme di R a causa di:

- **precisione** limitata
- **limitatezza** del dominio.

Lo spazio allocato per ogni numero reale (e quindi l'insieme dei valori rappresentabili) dipende dal metodo di rappresentazione adottato.

### Differenza tra float/double:

**float** *singola* precisione

**double** *doppia* precisione (maggiore numero di bit per la mantissa)

### Uso del quantificatore long:

si puo' applicare a **double**, per aumentare ulteriormente la precisione:

$spazio(\mathbf{float}) \leq spazio(\mathbf{double}) \leq spazio(\mathbf{long\ double})$

**Esempio:** definizione di variabili reali

```
float x;  
double A, B;
```

## Tipi float/double

### Operatori

#### Operatori aritmetici:

$+, -, *, /$  si applicano a operandi reali e producono risultati reali

#### Operatori relazionali:

hanno lo stesso significato visto nel caso degli interi:

$==, !=$  uguale, diverso

$<, >, <=, >=$  minore, maggiore etc.

#### Overloading:

Il C (come 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 od interi).

## Esempi:

5.0 / 2       $\Rightarrow$  2.5

2.1 / 2       $\Rightarrow$  1.05

7.1 < 4.55       $\Rightarrow$  0

17 == 121       $\Rightarrow$  0

$\Rightarrow$  A causa della rappresentazione finita, ci possono essere errori di conversione. Ad esempio, i test di uguaglianza tra valori reali (in teoria uguali) potrebbero non essere verificati.

$$(x / y) * y == x$$

Meglio utilizzare "un margine accettabile di errore":

$$(X == Y) \quad \Rightarrow \quad (X <= Y + \epsilon) \ \&\& \ (X <= Y - \epsilon)$$

dove, ad esempio:

**const float** epsilon=0.000001;

## Il tipo char

### Carattere:

ogni simbolo grafico rappresentabile all'interno del sistema. Ad esempio:

- le lettere dell'alfabeto (maiuscole, minuscole)
- le cifre decimali ('0'..'9')
- i segni di punteggiatura (';', '!' etc.)
- altri simboli di vario tipo ('+', '-', '&', '@', etc.).
- i caratteri di controllo (*bell*, *lf*, *,*, *ff*, etc.)

### Dominio del tipo char

E' l'insieme dei *caratteri* disponibili sul sistema di elaborazione (*set* di caratteri).

### Tabella dei Codici

Di solito, si fa riferimento ad una tabella dei codici (ad esempio: ASCII). In ogni tabella dei codici, ad ogni carattere viene associato un intero che lo identifica univocamente: il **codice**.

- Il dominio associato al tipo **char** e' **ordinato**: l'ordine dipende dal codice associato ai vari caratteri.

## Tabella ASCII

Di solito, vengono usati **8 bit** -> 256 valori possibili

0	NUL	42	*	84	T	126	~	168	®	210	“	252	
1	SOH	43	+	85	U	127	□	169	©	211	”	253	
2	STX	44	,	86	V	128	Ä	170	™	212	‘	254	
3	ETX	45	-	87	W	129	Å	171	’	213	’	255	
4	EOT	46	.	88	X	130	Ç	172	¨	214	÷		
5	ENQ	47	/	89	Y	131	É	173	≠	215	∅		
6	ACK	48	0	90	Z	132	Ñ	174	Æ	216	ÿ		
7	BEL	49	1	91	[	133	Ö	175	Ø	217	ÿ		
8	BS	50	2	92	\	134	Ü	176	∞	218	/		
9	HT	51	3	93	]	135	á	177	±	219	∓		
10	LF	52	4	94	^	136	à	178	≤	220	<		
11	VT	53	5	95	_	137	â	179	≥	221	>		
12	FF	54	6	96	`	138	ã	180	¥	222	fi		
13	CR	55	7	97	a	139	ä	181	µ	223	fl		
14	SO	56	8	98	b	140	å	182	∂	224	‡		
15	SI	57	9	99	c	141	ç	183	∑	225	·		
16	DLE	58	:	100	d	142	é	184	∏	226	,		
17	DC1	59	;	101	e	143	è	185	π	227	..		
18	DC2	60	<	102	f	144	ê	186	∫	228	%		
19	DC3	61	=	103	g	145	ë	187	ª	229	Å		
20	DC4	62	>	104	h	146	í	188	°	230	È		
21	NAK	63	?	105	i	147	ì	189	Ω	231	Á		
22	SYN	64	@	106	j	148	î	190	æ	232	Ë		
23	ETB	65	A	107	k	149	ï	191	ø	233	È		
24	Can	66	B	108	l	150	ñ	192	¿	234	Í		
25	EM	67	C	109	m	151	ó	193	¡	235	Ï		
26	SUB	68	D	110	n	152	ò	194	¢	236	Î		
27	ESC	69	E	111	o	153	ô	195	√	237	Ï		
28	FS	70	F	112	p	154	ö	196	f	238	Ó		
29	GS	71	G	113	q	155	ø	197	≈	239	Ô		
30	RS	72	H	114	r	156	ú	198	Δ	240	□		
31	US	73	I	115	s	157	ù	199	«	241	Ò		
32		74	J	116	t	158	û	200	»	242	Ú		
33	!	75	K	117	u	159	ü	201	...	243	Û		
34	"	76	L	118	v	160	ÿ	202		244	Ü		
35	#	77	M	119	w	161	°	203	À	245	ı		
36	\$	78	N	120	x	162	¢	204	Á	246	^		
37	%	79	O	121	y	163	£	205	Ï	247	~		
38	&	80	P	122	z	164	§	206	Œ	248	—		
39	'	81	Q	123	{	165	•	207	œ	249	˘		
40	(	82	R	124		166	¶	208	–	250	˙		
41	)	83	S	125	}	167	ß	209	—	251	°		

## Il tipo char

Il dominio associato al tipo **char** è **ordinato**: l'ordine dipende dal codice associato ai vari caratteri nella tabella di riferimento.

### Definizione di variabili di tipo char: esempio

**char C1, C2;**

### Costanti di tipo char:

Ogni valore di tipo char viene specificato tra singoli apici.

### Ad esempio:

'a' 'b' 'A' '0' '2'

### Rappresentazione dei caratteri in C:

Il linguaggio C rappresenta i dati di tipo **char** come degli **interi**:

ogni carattere viene rappresentato dal suo codice (cioè, **l'intero** che lo indica nella tabella ASCII)

# Special Characters

- Escape sequences are used to represent many special characters in C

<code>\n</code>	newline	each escape sequence represents only one character
<code>\t</code>	tab	
<code>\b</code>	backspace	
<code>\f</code>	form feed	
<code>\a</code>	audible bell	
<code>\0</code>	null	

- Can be mixed freely with other characters

```
printf("\nA\nB\tC\nDE\aF\n"); /* what does this print? */
```

do NOT try to print the null (`'\0'`) character

## Il tipo char: Operatori

### I char sono rappresentati da interi (su 8 bit):

☞ sui dati **char** è possibile eseguire tutte le operazioni previste per gli interi. Ogni operazione, infatti, è applicata ai codici associati agli operandi.

### Operatori relazionali:

`==, !=, <, <=, >=, >` per i quali valgono le stesse regole viste per gli interi

### Ad esempio:

`char x,y;`  
`x < y` se e solo se `codice(x) < codice(y)`

`'a' > 'b'` *false* perché `codice('a') < codice('b')`

### Operatori aritmetici:

sono gli stessi visti per gli interi.

### Operatori logici:

sono gli stessi visti per gli interi.

### Esempi:

`'A' < 'C'`       $\Rightarrow$  **1** (infatti `65 < 67` e' vero)

`' "' + '!''`  $\Rightarrow$  `'C'` (`codice("")+codice(!)=67`)

`!'A'`       $\Rightarrow$  **0** (`codice(A)` e' diverso da zero)

`'A' && 'a'`  $\Rightarrow$  **1**

### Uso dei qualificatori:

e' possibile, come per gli interi, applicare i qualificatori `signed`, `unsigned` a variabili di tipo `char`:

```
signed char C;  
unsigned char K;
```

# Data Types: Int vs Char

Why is it that given ...

```
char number1;  
int  number2;  
int  number3;  
  
number1 = 160;           /* assigns values */  
number2 = 565;           /* to variables */  
number3 = number2;  
number2 = number1;      /* this is OK */  
number1 = number3;      /* this is NOT? */
```

# COSTANTI E DICHIARAZIONI DI COSTANTI

## Oltre alle Costanti esplicite

esprimono direttamente dei valori

23	costante di tipo int
3.1416	costante di tipo double
'A'	costante di tipo char

## Ci sono le Costanti simboliche

- sono **nomi** simbolici che il programmatore adotta convenzionalmente per indicare dei valori prefissati
- hanno un **tipo** espresso implicitamente dal valore

## La dichiarazione di una costante

- definisce il nome
- associa un valore (e tipo implicito)

*In C la «dichiarazione» (definizione) di costante può essere fatta tramite **direttiva al preprocessore C***

```
#define nmaxp      10
#define vmax      150.0
#define FALSE     0
#define TRUE      1
```

*In compilazione, viene effettuata una **sostituzione letterale** del nome simbolico con il valore associato.*

E' anche possibile una **dichiarazione** del tipo:

```
const <tipo> <nome> = <valore>;
```



# Symbolic Constants

---

```
const int max_buffer = 512;
```

```
const float e_approx = 2.718f;
```

- Much preferred to “magic” numbers
  - \* What does “512” mean?
  - \* Localized if need to change value
  - \* What if “512” means a number of things?!
- Memory is allocated
- Must be initialized (because ...)
- Cannot be changed, i.e., is “read-only”
- Preferred to `#defines`

## ISTRUZIONI DI ASSEGNAIMENTO

Indicano l'**operazione** che assegna un **valore** ad una **variabile**

### Sintassi C:

```
<nome_variabile> = <espressione>;
```

- <nome\_variabile> indica il nome della **variabile** a cui si vuole assegnare un valore, in sostituzione di quello precedentemente contenuto in essa ... **è un indirizzo**
- = è il simbolo di assegnamento
- <espressione> descrive come ottenere il **valore** da assegnare alla variabile

### Significato (semantica):

si eseguono le operazioni descritte in <espressione>

il valore ottenuto viene inserito nella posizione di memoria indicata dalla variabile a sinistra del simbolo di assegnamento.

**Compatibilità tra i tipi:** il compilatore controlla la compatibilità tra tipi. In alcune situazioni, risolve la non compatibilità adottando delle regole di conversione implicita e automatica tra tipi.

- il valore generato da <espressione> dovrebbe essere dello stesso tipo della variabile da assegnare
- assegnamento tra tipi eterogenei (tipi aritmetici): all'atto dell'assegnamento, il valore di <espressione> viene «convertito» in un valore corrispondente appartenente al tipo della variabile da assegnare.

## ESPRESSIONI E OPERATORI

### Sintassi C

<espressione>: contiene identificatori (di variabili, di costanti, di funzioni), costanti esplicite, operatori, ( )

### Semantica

descrive il modo con cui ottenere dal valore degli operandi e dall'applicazione degli operatori (operazioni) il valore dell'espressione.

Nelle espressioni complesse la **sequenza di esecuzione delle operazioni** è dettata dalla

- **precedenza** predefinita degli operatori
- **forzatura** mediante l'uso delle parentesi tonde

### Operatori

- unari si applicano ad un solo operando
- binari si applicano a due operandi



# Operatori aritmetici

- `++` → incremento unitario
- `--` → decremento unitario
- `*` → moltiplicazione
- `/` → divisione
- `%` → resto di una divisione fra interi
- `+` → addizione
- `-` → sottrazione

# Esempi

- ```
foo1() {  
    int x=0, y= 10;  
        x = y ++;           x = ++ y;  
}
```

valore → x=10, y=11      x=12, y=12 (*da sx verso dx*)
- ```
foo2() {  
    int x=16, y= 5, z=0;  
        z = x / y;         z = x % y;  
}
```

valore → z=3      z=1

## Booleani

Sono dati il cui dominio e` di due soli valori (valori logici):

$\{vero, falso\}$

☞ in C **non esiste** un tipo primitivo per rappresentare dati booleani.

### Come vengono rappresentati i risultati di espressioni relazionali ?

Il C prevede che i valori logici restituiti da espressioni relazionali vengano rappresentati attraverso gli interi  $\{0,1\}$  secondo la convenzione:

- **0 equivale a falso**
- **1 equivale a vero**

### Ad esempio:

l'espressione  $A == B$  restituisce:

- ☞ **0**, se la relazione non e` vera
- ☞ **1**, se la relazione e` vera

## Operatori logici:

si applicano ad operandi di tipo **int** e producono risultati *booleani*, cioe` interi appartenenti all'insieme  $\{0,1\}$  (il valore 0 corrisponde a "falso", il valore 1 corrisponde a "vero"). In particolare l'insieme degli operatori logici e`:

<b>&amp;&amp;</b>	operatore AND logico
<b>  </b>	operatore OR logico
<b>!</b>	operatore di negazione (NOT)

### Definizione degli operatori logici:

<b>a</b>	<b>b</b>	<b>a&amp;&amp;b</b>	<b>a  b</b>	<b>!a</b>
<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>vero</i>	<i>vero</i>
<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>	<i>falso</i>
<i>vero</i>	<i>vero</i>	<i>vero</i>	<i>vero</i>	<i>falso</i>

## Operatori Logici in C

**In C, gli operandi di operatori logici sono di tipo int:**

- se il valore di un operando e' **diverso da zero**, viene interpretato come *vero*.
- se il valore di un operando e' **uguale a zero**, viene interpretato come *falso*.

### Definizione degli operatori logici in C:

a	b	a&&b	a  b	!a
0	0	0	0	1
0	≠ 0	0	1	1
≠ 0	0	0	1	0
≠ 0	≠ 0	1	1	0

## Esempi sugli operatori tra interi:

37 / 3       $\Rightarrow$  12

37 % 3       $\Rightarrow$  1

7 < 3       $\Rightarrow$  0

7 >= 3       $\Rightarrow$  1

0 || 1       $\Rightarrow$  1

0 || -123       $\Rightarrow$  1

12 && 2       $\Rightarrow$  1

0 && 17       $\Rightarrow$  0

!2       $\Rightarrow$  0



## ...e anche ... **Operatori sui Bit**

■  $\sim$  → complemento a uno

■  $\ll$  → scorrimento a sinistra

■  $\gg$  → scorrimento a destra

■  $\&$  → **AND**

■  $\wedge$  → XOR

	0	0	1	1
	0	1	1	0
XOR	0	1	0	1

■  $|$  → **OR**



# Logical and Bitwise Operators

---

- Logical (true/false): `&&`, `||`, `!`

```
int n = 7 || 0;    // n == 1
```

- One's complement: `~`

```
n = ~4;    // all bits on except third to last
```

- Bitwise shifts: `<<`, `>>`

```
n = 12 >> 2;    // n == 3
```

- \* standard set for unsigned, integral types only

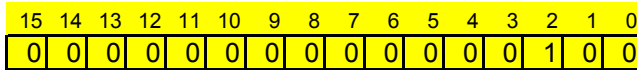
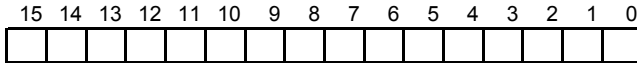
- \* bit-wrap, 0-fill or 1-fill is direction and compiler dependent

- Bitwise masks, logically bit-by-bit: `&`, `|`, `^`

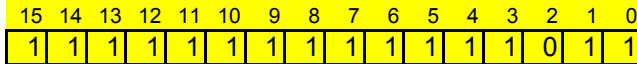
```
n = n & ~017;    // zeros out last 4 bits
```

Note: better than `n & 0177760` which assumes  $\geq 16$  bits

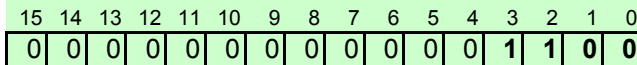
es su 16 bit



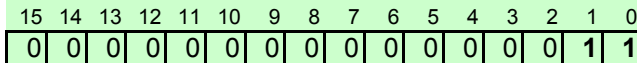
4



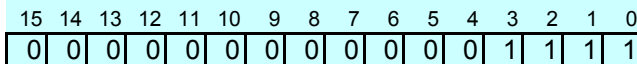
$n = \sim 4$



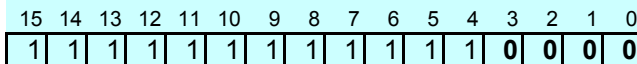
12



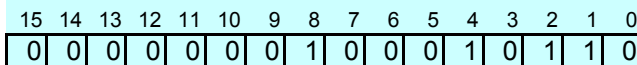
$n = 12 \gg 2$



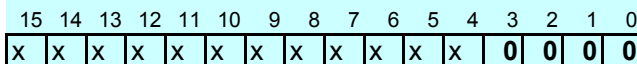
017



$\sim 017$



$n$  qualsiasi



$n = n \& \sim 017$



# Esempi

- foo1() {

```
int x=0, y= 0, z=29;
```

→ z=0001 1101

```
    x = z << 1;
```

→ x=0011 1010

```
    x = z >> 1;
```

→ x=0000 1110

```
    y = ~z; }
```

→ y=1110 0010

- foo2() {

```
int x=3, y= 5, z=0;
```

→ x=0011, y=0101

```
    z = x & y;
```

→ z=0001

```
    z = x | y; }
```

→ z=0111

## Regole di Precedenza e Associativita' degli Operatori C (in ordine di priorit  decrescente)

Operatore	Associativita'
( ) [ ] ->	da sinistra a destra
! ~ ++ -- & sizeof	da destra a sinistra
* / %	da sinistra a destra
+ -	da sinistra a destra
<< >>	da sinistra a destra
< <= > >=	da sinistra a destra
== !=	da destra a sinistra
&	da sinistra a destra
^	da sinistra a destra
	da sinistra a destra
&&	da sinistra a destra
	da sinistra a destra
+= -= *= /=	da destra a sinistra

## Precedenza e Associativita'

### Esempi

$3*5 \% 2$   $\implies$  equivale a:  $(3*5) \% 2$   
 $X+7-A$   $\implies$  equivale a:  $(X+7) - A$   
 $3 < 0 \ \&\& \ 3 < 10$   $\implies (3 < 0) \ \&\& \ (3 < 10)$   $\implies 0 \ \&\& \ 1$   
 $3 < (0 \ \&\& \ 3) < 10$   $\implies (3 < 0) < 10$   $\implies 0 < 1$   
 $0 == 7 == 3$   $\implies 0 == (7 == 3)$   $\implies 0 == 0$

### Valutazione a "corto circuito" (*short-cut*):

nella valutazione di una espressione C, se un risultato intermedio determina a priori il risultato finale della espressione, il resto dell'espressione non viene valutato.

### Ad esempio, espressioni logiche:

#### Hp. Valutazione degli operandi da sin a destra

$(3 > 0) \ \&\& \ (X < Y)$   $\implies$  solo primo operando  
**falso** && **vero**

☞ Bisognerebbe evitare di scrivere espressioni che dipendono dal metodo di valutazione usato  $\implies$  scarsa portabilit  (ad es., in presenza di funzioni con effetti collaterali).