




# **I tipi di dato fondamentali**

Francesco Isgrò

- 
- Richiamiamo dichiarazioni, espressioni e assegnazioni
  - Vediamo in dettaglio i tipi di dato fondamentali
  - In particolare interi e caratteri
  - Conversione implicita
  - Operatore di cast

# Tipi di dato fondamentali

Fundamental data types: long form		
char	signed char	unsigned char
signed short int	signed int	signed long int
unsigned short int	unsigned int	unsigned long int
float	double	long double



Fundamental data types: long form		
char	signed char	unsigned char
signed short int	signed int	signed long int
unsigned short int	unsigned int	unsigned long int
float	double	long double

- Sono tutti parole chiave
- Non possono essere usati come nomi di variabili
- La parola signed solitamente non è usata
- signed int è equivalente a int
- short int, long int e unsigned int solitamente abbreviate
- signed da solo equivalente a int, ma raramente usato

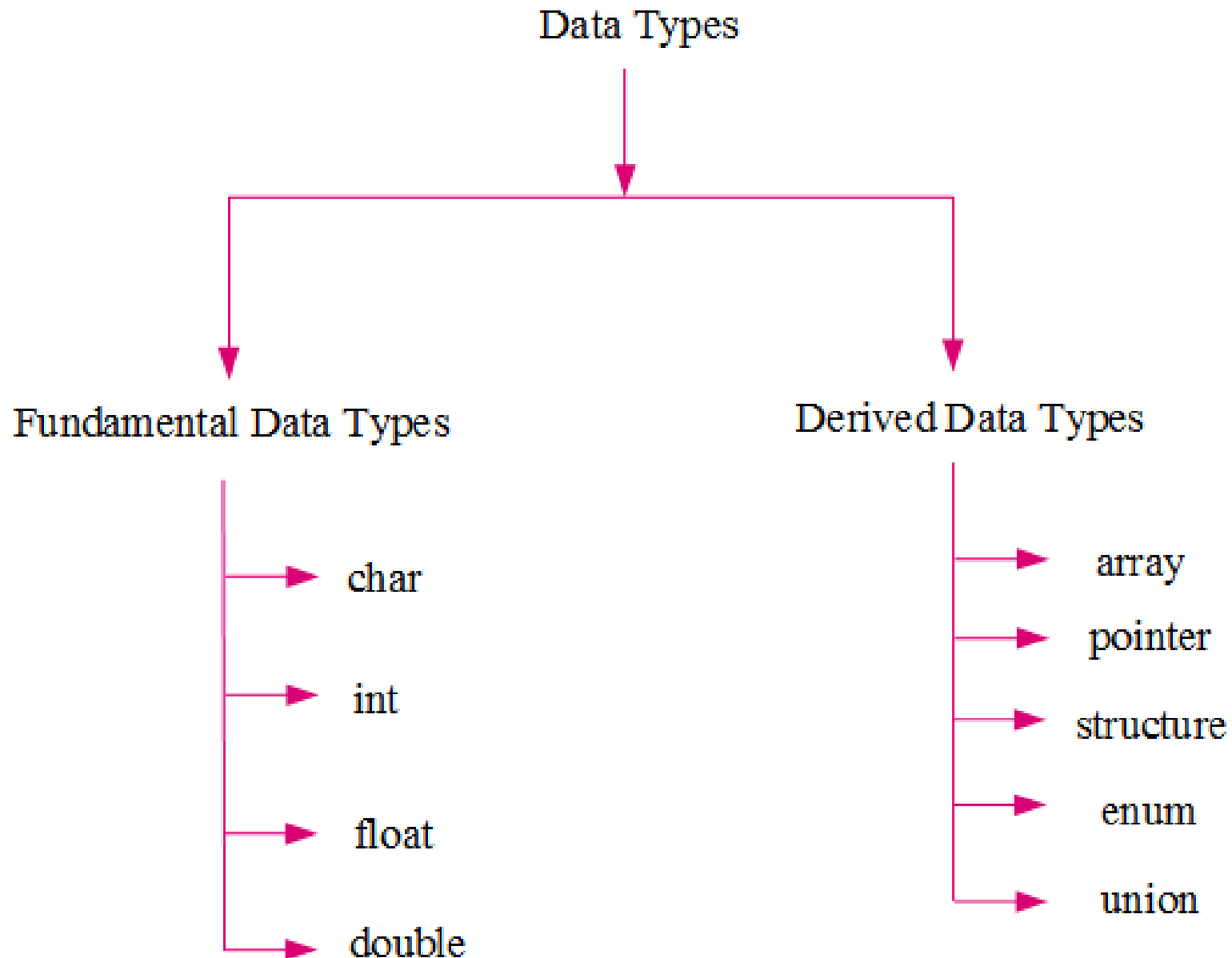


Fundamental data types		
char	signed char	unsigned char
short	int	long
unsigned short	unsigned	unsigned long
float	double	long double



Fundamental data types grouped by functionality			
Integral types	char	signed char	unsigned char
	short	int	long
	unsigned short	unsigned	unsigned long
Floating types	float	double	long double
Arithmetic types	<i>Integral types + Floating types</i>		


# Tipi di dato derivati



# Caratteri e il tipo char

- In C ogni tipo intero può essere usato per rappresentare caratteri
- In particolare sia char sia int sono usati a questo scopo
  - Funzioni come putchar() e getchar() usano int
  - Costanti come 'a' sono di tipo int
  - Non ci sono costanti di tipo char in C
  - In C++ costanti di caratteri sono di tipo char



- 
- Variabili di tipo char possono rappresentare valori interi piccoli
  - Un char è memorizzato in 1 byte
  - $2^8 = 256$  valori distinti
  - Solo un sottoinsieme di questi valori rappresenta caratteri stampabili
    - Maiuscole, minuscole, cifre, punteggiatura, caratteri speciali
    - Spazio bianco, tabulazione, newline



# Caratteri ASCII

- La maggior parte delle macchine usano la codifica ASCII
- Altra codifica, meno usata, è la EBCDIC dovuta a IBM


# Caratteri ASCII


- La maggior parte delle macchine usano la codifica ASCII
- Altra codifica, meno usata, è la EBCDIC dovuta a IBM
- Corrispondenza numero carattere
- Nessuna relazione nelle corrispondenze
- I valori seguono l'ordinamento naturale
  - 'a' < 'b' < 'c' < ..... < 'z'
  - Conveniente per ordinare stringhe di testo
- Alcuni caratteri speciali sono rappresentati da una sequenza


Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
1	01	Start of heading	33	21	!	65	41	A	97	61	a	129	81	ù	161	A1	í	193	C1	ł	225	E1	β
2	02	Start of text	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	ṽ	226	E2	Γ
3	03	End of text	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	ṭ	227	E3	π
4	04	End of transmit	36	24	\$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
5	05	Enquiry	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
6	06	Acknowledge	38	26	&	70	46	F	102	66	f	134	86	ã	166	A6	ª	198	C6	‡	230	E6	μ
7	07	Audible bell	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	‖	231	E7	τ
8	08	Backspace	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	⋮	232	E8	Φ
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	¬	201	C9	⋯	233	E9	Θ
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	⋮	234	EA	Ω
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	⋮	235	EB	Θ
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¾	204	CC	‖	236	EC	∞
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	=	237	ED	∞
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n	142	8E	Ë	174	AE	«	206	CE	‡	238	EE	ε
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o	143	8F	Ä	175	AF	»	207	CF	⋮	239	EF	Π
16	10	Data link escape	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	☐	208	DO	⋮	240	FO	≡
17	11	Device control 1	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	☐	209	D1	⋮	241	F1	±
18	12	Device control 2	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	☐	210	D2	⋮	242	F2	≥
19	13	Device control 3	51	33	3	83	53	S	115	73	s	147	93	ó	179	B3		211	D3	⋮	243	F3	≤
20	14	Device control 4	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	†	212	D4	⋮	244	F4	[
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	‡	213	D5	⋮	245	F5	]
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	‖	214	D6	⋮	246	F6	÷
23	17	End trans. block	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	⋮	215	D7	‡	247	F7	≈
24	18	Cancel	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	¶	216	D8	‡	248	F8	•
25	19	End of medium	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	‖	217	D9	¶	249	F9	•
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	‖	218	DA	¶	250	FA	•
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{	155	9B	÷	187	BB	¶	219	DB	■	251	FB	√
28	1C	File separator	60	3C	<	92	5C	\	124	7C		156	9C	£	188	BC	¶	220	DC	■	252	FC	Δ
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	¶	221	DD	■	253	FD	z
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~	158	9E	℔	190	BE	¶	222	DE	■	254	FE	■
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□	159	9F	ƒ	191	BF	¶	223	DF	■	255	FF	□



Special caharacters		
Name of character	Written in C	Integer value
alert	\a	7
backslash	\\	92
backspace	\b	8
Carriage return	\r	13
Double quote	\"	34
formfeed	\f	12
Horizzontal tab	\t	9
newline	\n	10
Null characther	\0	0
Single quote	\'	39
Vertical tab	\v	11
Question mark	\?	63

- 
- Alert : \a
    - Bell ring
    - `printf(“%c”\a’)`
  - Double quote
    - Necessario se dobbiamo scrivere “ in una stringa costante
    - `printf(“Di nome %cNemo%c”,\”,\”);`
    - Stampa  
Di nome “Nemo”

- 
- Carriage return
    - `\r`
    - Fa tornare il cursore all'inizio
    - Come nelle macchine da scrivere: accapo era la sequenza `\r\n`




```
#include <stdio.h>
```

```
int main(void)
{
    printf("abc \t abc\n");
    printf("abc \v abc\n");
    printf("abc \f abc\n");
    printf("abc \r abc\n");
    printf("abc \r \v abc\n");

    return 0;
}
```






```
#include <stdio.h>
```

```
int main(void)
{
    printf("abc \t abc\n");
    printf("abc \v abc\n");
    printf("abc \f abc\n");
    printf("abc \r abc\n");
    printf("abc \r \v abc\n");

    return 0;
}
```

```
abc  abc
abc
  abc
abc
  abc
abc
bc
abc
```




```
#include <stdio.h>
```


```
#define PAUSA 2000000
```

```
void pausa(int l)
{
    int i;
    for (i=0; i<l; i++);
}
```

```
int main()
{
    setbuf(stdout, 0);
    while(1) {
        printf("|\\r"); pausa(PAUSA);
        printf("/\\r"); pausa(PAUSA);
        printf("-\\r"); pausa(PAUSA);
        printf("\\\\r"); pausa(PAUSA);

    }
}
```

- 
- Vediamo come
    - Caratteri sono trattati come interi piccoli
    - Interi piccoli sono trattati come caratteri
    - `Char c = 'a';`
    - `printf(“%c”, c);`
    - `printf(“%d”, c);`

- 
- Vediamo come
    - Caratteri sono trattati come interi piccoli
    - Interi piccoli sono trattati come caratteri

```
char c = 'a';
```

```
printf("%c\n", c);
```

```
printf("%d\n", c);
```

- Viene stampato

a

97


- 
- Siccome una variabile char ha un valore intero si può usare in espressioni aritmetiche

```
printf("%c%c%c\n", c, c+1, c+2);
```

- 
- Siccome una variabile char ha un valore intero si può usare in espressioni aritmetiche

```
printf("%c%c%c\n", c, c+1, c+2);
```

- Produce  
abc




```
int i;  
char c;
```


```
//stampa abc....z  
for (i='a'; i<='z'; i++)  
    printf("%c", i);
```

```
//stampa ABC.....Z  
for (c=65; c<=90; c++)  
    printf("%c",c);
```

```
//stampa 48 49 .... 57  
for (c='0'; c<='9'; c++)  
    printf("%d ", c);
```

- 
- Attenzione ad usare gli operatori ++ e –
  - A volte i risultati non sono prevedibili





```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char c; int a;
```

```
    c = 'a'; a=10;
```

```
    printf("%c%c%c\n", c, c, c++);
```

```
    printf ("%d %d %d \n", a , a , a++);
```

```
    c = 'a'; a=10;
```

```
    printf("%c%c%c\n", c, c, ++c);
```

```
    printf("%d %d %d \n", a , a , ++a);
```

```
    c = 'a'; a=10;
```

```
    printf("%c%c%c\n", c, ++c, ++c);
```

```
    printf("%d %d %d \n", a , ++a , ++a);
```

```
    c = 'a'; a=10;
```

```
    printf("%c%c%c\n", c, c++, c++);
```


```
    printf("%d %d %d \n", a , a++ , a++);
```

```
    c = 'a'; a=10;
```

```
    printf("%c%c%c \n", ++c , c++ , c);
```

```
    printf("%d %d %d \n", ++a , a++ , a);
```

```
}
```



```
#include <stdio.h>
```

```
int main()
```

```
{  
    char c; int a;
```

```
    c = 'a'; a=10;  
    printf("%c%c%c\n", c, c, c++);  
    printf ("%d %d %d \n", a , a , a++);
```

```
    c = 'a'; a=10;  
    printf("%c%c%c\n", c, c, ++c);  
    printf("%d %d %d \n", a , a , ++a);
```

```
    c = 'a'; a=10;  
    printf("%c%c%c\n", c, ++c, ++c);  
    printf("%d %d %d \n", a , ++a , ++a);
```


```
    c = 'a'; a=10;  
    printf("%c%c%c\n", c, c++, c++);  
    printf("%d %d %d \n", a , a++ , a++);
```

```
    c = 'a'; a=10;  
    printf("%c%c%c \n", ++c , c++ , c);  
    printf("%d %d %d \n", ++a , a++ , a);  
}
```

```
bba  
11 11 10  
bbb  
11 11 11  
ccb  
12 12 12  
cba  
12 11 10  
caa  
12 10 12
```

# Rappresentazione binaria di un char

- Il sistema binario è posizionale come il decimale
- Nel sistema decimale
  - $20753 = 2*10^4 + 0*10^3 + 7*10^2 + 5*10^1 + 3*10^0$
  - 3 è la cifra meno significativa ( $10^0$ )
  - 2 è la cifra più significativa ( $10^4$ )
- La forma generale è
  - $d_n d_{n-1} \dots d_2 d_1 d_0$
  - Ogni  $d_i$  è una cifra
  - Il valore del numero è dato da
  - $d_n * 10^n + d_{n-1} * 10^{n-1} \dots + d_2 * 10^2 + d_1 * 10^1 + d_0 * 10^0$

- 
- Analogamente una sequenza di 0 e 1 può essere interpretata come un numero binario
  - La forma generale è
    - $b_n b_{n-1} \dots b_2 b_1 b_0$
    - Ogni  $b_i$  è un binary digit (0 o 1)
    - Il valore è dato da
    - $b_n * 2^n + b_{n-1} * 2^{n-1} \dots + b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0$

- 
- Data la dichiarazione


`char c = 'a';`


- La variabile c è conservata in memoria in un byte come

01100001 =

$$0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$

$$64 + 32 + 1 = 97$$

- 
- Il C fornisce i tre tipi
    - char
    - signed char
    - unsigned char
  - char dipende dall'implementazione
    - Equivalente a signed char
    - Equivalente a unsigned char

- 
- Ognuno dei tre tipi occupa 1 byte di memoria
    - 256 valori distinti
    - Signed char va da -128 a 127
    - Unsigned da 0 a 255


# Valori negativi


- Come vengono rappresentati i valori negativi?
- L'idea più semplice è il bit di segno. Limitiamoci a 4 bit per semplicità
  - Il bit più significativo è riservato al segno
    - $5 = 0101$
    - $-5 = 1101$



# Valori negativi

- Come vengono rappresentati i valori negativi?
- L'idea più semplice è il bit di segno. Limitiamoci a 4 bit per semplicità
  - Il bit più significativo è riservato al segno
    - $5 = 0101$
    - $-5 = 1101$
  - Con lo 0
    - $0 = 0000$
    - $0 = 1000$
    - 2 rappresentazioni: 1 valore in meno rappresentato

- 
- Solitamente si usa il complemento a 2
    - I numeri positivi si rappresentano normalmente
    - Per i negativi:
      - Negata rappresentazione positivo
      - Si somma 1
    - In questa maniera 1000 non rappresenta 0 ma -8


- 
- Calcoliamo la rappresentazione di -5
    - $5 = 0101$
    - La negazione è 1010
    - Sommando 1 ( $1010 + 0001$ ) si ha 1011



-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

# Determinare range di char

- Char dipende dall'implementazione
  - Signed
  - Unsigned
- Come si fa a capire come è implementato?



```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char    c = -1;
```


```
    signed char  s = -1;
```

```
    unsigned char u = -1;
```

```
    printf("c = %d  s = %d  u = %d \n", c, s, u);
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>
```

```
int main()
{
    char      c = -1;
    signed char s = -1;
    unsigned char u = -1;

    printf("c = %d  s = %d  u = %d \n", c, s, u);

    return 0;
}
```

- Ognuna delle variabili è in memoria come 11111111
- Se char è signed viene stampato  
 $c = -1 \ s = -1 \ u = 255$
- Se char è unsigned viene stampato  
 $c = 255 \ s = -1 \ u = 255$

# Il tipo int

- Il tipo principale del linguaggio C
- Solitamente un int è conservato in 2 byte (16 bit) o in 4 byte (32 bit) di memoria
  - 16 bit in vecchi sistemi
  - 32 bit quasi sempre in sistemi moderni
  - La quantità di valori distinti quindi dipende dal sistema



- 
- Con 16 bit il range dei valori è

$$-2^{15}, -2^{15}+1, \dots, -2, -1, 0, 1, 2, \dots, 2^{15}-1$$

- Con 32 bit il range è

$$-2^{31}, -2^{31}+1, \dots, -2, -1, 0, 1, 2, \dots, 2^{31}-1$$

- Indichiamo con
  - $N_{\text{max\_int}}$  il più grande intero che può essere rappresentato
  - $N_{\text{min\_int}}$  il più piccolo intero che può essere rappresentato
  - Una qualsiasi variabile int  $i$  è tale che
  - $N_{\text{min\_int}} \leq i \leq N_{\text{max\_int}}$

- 
- In un sistema con interi a 16 bit


- $N_{\text{min\_int}} = -2^{15} = -32768$

- $N_{\text{max\_int}} = 2^{15} - 1 = 32767$

- In un sistema con interi a 32 bit

- $N_{\text{min\_int}} = -2^{31} = -2147483648$  circa -2 miliardi

- $N_{\text{max\_int}} = 2^{31} - 1 = 2147483647$  circa 2 miliardi



```
#define BIG 2000000000 // 2 miliardi
```

```
int main(void)
```

```
{
```

```
    int a, b = BIG, c = BIG;
```

```
    a = b + c;
```

```
    .....
```

```
}
```



```
#define BIG 2000000000 // 2 miliardi
```

```
int main(void)
{
    int a, b = BIG, c = BIG;

    a = b + c;

    .....
}
```

- Il codice è sintatticamente corretto
- In esecuzione
  - Il valore di  $a = b + c > N_{\text{max\_int}}$
  - Condizione di integer overflow
  - Il programma continua l'esecuzione ma i risultati sono incorretti
  - Bisogna prestare attenzione che i valori siano nell'intervallo appropriato



# short, long e unsigned

- Oltre int e char sono gli altri integral types
- Introdotti per usi più specializzati
  - Solitamente short usano meno spazio di int
  - short quando lo spazio in memoria è importante
  - long se si devono usare valori grandi

- 
- Solitamente short è 2 byte


- $N_{\text{min\_int}} = -2^{15} = -32768$

- $N_{\text{max\_int}} = 2^{15} - 1 = 32767$


- Solitamente long è 8 byte

- $N_{\text{min\_int}} = -2^{63} = -9223372036854775808$

- $N_{\text{max\_int}} = 2^{63} - 1 = 9223372036854775807$


- 
- Unsigned è lo stesso di int, tipicamente 4 byte
    - $N_{\text{min\_int}} = 0$
    - $N_{\text{max\_int}} = 2^{32}-1 = 4294967295$





- 
- A una costante intera è assegnato il primo tipo in grado di contenerlo fra
    - int
    - long
    - unsigned
  - Il tipo si può forzare usando un suffisso
    - 37U      unsigned
    - 37L      long
    - 37UL     unsigned long

# Tipi reali

- ANSI C fornisce tre tipi reali
  - float
  - double
  - long
- Le costanti reali sono di tipo double
  - Il tipo si può forzare con un suffisso
    - 3.7F float
    - 3.7L long double

- 
- Solitamente il compilatore fornisce più memoria ai double che ai float
    - Non è obbligatorio
    - Su molte macchine
    - float      4 byte
    - double    8 byte
  - Il compilatore può fornire più memoria ai long double che ai double
    - Non obbligatorio


- 
- I possibili valori assegnabili sono descritti da
    - *precisione*: descrive il numero di cifre significative rappresentabili in notazione decimale
    - *range*: descrive quali siano il più grande e il più piccolo valore rappresentabile

- 
- Su molte macchine un float ha
    - precisione di 6 cifre significative
    - range fra  $10^{-38}$  e  $10^{+38}$
  - Si può quindi rappresentare nella forma

$$0.d_1d_2d_3d_4d_5d_6 * 10^n$$

dove


- di  $d_i$  è una cifra decimale
- $-38 \leq n \leq 38$

- 
- Su molte macchine un float ha
    - precisione di 6 cifre significative
    - range fra  $10^{-38}$  e  $10^{+38}$
  - Si può quindi rappresentare nella forma

$$0.d_1d_2d_3d_4d_5d_6 * 10^n$$

dove

- di  $d_i$  è una cifra decimale
  - $-38 \leq n \leq 38$
- In realtà si usa la base 2, non 10 nella rappresentazione sulla macchina




```
int main()
{
    float x;

    x = 1e-37; printf("x = 1e-37    x = %e\n", x);
    x = 1e-38; printf("x = 1e-38    x = %e\n", x);
    x = 1e-39; printf("x = 1e-39    x = %e\n", x);
    x = 1e-40; printf("x = 1e-40    x = %e\n", x);
    x = 1e-41; printf("x = 1e-41    x = %e\n", x);
    x = 1e-42; printf("x = 1e-42    x = %e\n", x);
    x = 1e-43; printf("x = 1e-43    x = %e\n", x);
    x = 1e-44; printf("x = 1e-44    x = %e\n", x);
    x = 1e-45; printf("x = 1e-45    x = %e\n", x);
    x = 1e-46; printf("x = 1e-46    x = %e\n", x);

    x = 1e38; printf("x = 1e38    x = %e\n", x);
    x = 1e39; printf("x = 1e39    x = %e\n", x);

    x = 123.45123451234512345;
    printf("x = 123.45123451234512345    x = %3.17f\n", x);
}
```




x = 1e-37   x = 1.000000e-37  
x = 1e-38   x = 9.999999e-39  
x = 1e-39   x = 1.000000e-39  
x = 1e-40   x = 9.999946e-41  
x = 1e-41   x = 9.999666e-42  
x = 1e-42   x = 1.000527e-42  
x = 1e-43   x = 9.949219e-44  
x = 1e-44   x = 9.809089e-45  
x = 1e-45   x = 1.401298e-45  
x = 1e-46   x = 0.000000e+00

x = 1e38   x = 1.000000e+38  
x = 1e39   x = inf

x = 123.45123451234512345   x = 123.45123291015625000






```
int main()
{
    float x;

    x = 1e-37; printf("x = 1e-37    x = %e\n", x);
    x = 1e-38; printf("x = 1e-38    x = %e\n", x);
    x = 1e-39; printf("x = 1e-39    x = %e\n", x);
    x = 1e-40; printf("x = 1e-40    x = %e\n", x);
    x = 1e-41; printf("x = 1e-41    x = %e\n", x);
    x = 1e-42; printf("x = 1e-42    x = %e\n", x);
    x = 1e-43; printf("x = 1e-43    x = %e\n", x);
    x = 1e-44; printf("x = 1e-44    x = %e\n", x);
    x = 1e-45; printf("x = 1e-45    x = %e\n", x);
    x = 1e-46; printf("x = 1e-46    x = %e\n", x);

    x = 1e38; printf("x = 1e38    x = %e\n", x);
    x = 1e39; printf("x = 1e39    x = %e\n", x);


    x = 123.45123451234512345;
    printf("x = 123.45123451234512345    x = %e\n", x);
}
```



x = 1e-37   x = 1.000000e-37  
x = 1e-38   x = 9.999999e-39  
x = 1e-39   x = 1.000000e-39  
x = 1e-40   x = 9.999946e-41  
x = 1e-41   x = 9.999666e-42  
x = 1e-42   x = 1.000527e-42  
x = 1e-43   x = 9.949219e-44  
x = 1e-44   x = 9.809089e-45  
x = 1e-45   x = 1.401298e-45  
x = 1e-46   x = 0.000000e+00


x = 1e38   x = 1.000000e+38  
x = 1e39   x = inf

x = 123.451251234512345   x = 1.234512e+02

- 
- Su molte macchine un double ha
    - precisione di circa 15 cifre
    - range compreso fra  $10^{-308}$  e  $10^{+308}$
  - Si può rappresentare nella forma
    - $0.d_1d_2d_3 \dots d_{14}d_{15} * 10^n$

dove

- di  $d_i$  è una cifra decimale
- $-308 \leq n \leq 308$



```
#include <stdio.h>
```

```
int main()  
{  
    double x;
```

```
    x = 1e-307; printf("x = 1e-307    x = %e\n", x);  
    x = 1e-308; printf("x = 1e-308    x = %e\n", x);  
    x = 1e-309; printf("x = 1e-309    x = %e\n", x);  
    x = 1e-400; printf("x = 1e-400    x = %e\n", x);  
    x = 1e-401; printf("x = 1e-401    x = %e\n", x);  
    x = 1e-402; printf("x = 1e-402    x = %e\n", x);  
    x = 1e-403; printf("x = 1e-403    x = %e\n", x);  
    x = 1e-404; printf("x = 1e-404    x = %e\n", x);  
    x = 1e-405; printf("x = 1e-405    x = %e\n", x);  
    x = 1e-406; printf("x = 1e-406    x = %e\n", x);
```

```
    x = 1e308; printf("x = 1e308    x = %e\n", x);  
    x = 1e309; printf("x = 1e309    x = %e\n", x);
```

```
    x = 123.45123451234512345;  
    printf("x = 123.45123451234512345    x = %e\n", x);  
}
```



double.c: In function 'main':

double.c:10:4: warning: floating constant truncated to zero [-Woverflow]

```
x = 1e-400; printf("x = 1e-400  x = %e\n", x);  
^
```

double.c:11:4: warning: floating constant truncated to zero [-Woverflow]

```
x = 1e-401; printf("x = 1e-401  x = %e\n", x);  
^
```

double.c:12:4: warning: floating constant truncated to zero [-Woverflow]

```
x = 1e-402; printf("x = 1e-402  x = %e\n", x);  
^
```

double.c:13:4: warning: floating constant truncated to zero [-Woverflow]

```
x = 1e-403; printf("x = 1e-403  x = %e\n", x);  
^
```

double.c:14:4: warning: floating constant truncated to zero [-Woverflow]

```
x = 1e-404; printf("x = 1e-404  x = %e\n", x);  
^
```

double.c:15:4: warning: floating constant truncated to zero [-Woverflow]


```
x = 1e-405; printf("x = 1e-405  x = %e\n", x);  
^
```

double.c:16:4: warning: floating constant truncated to zero [-Woverflow]

```
x = 1e-406; printf("x = 1e-406  x = %e\n", x);  
^
```

double.c:19:4: warning: floating constant exceeds range of 'double' [-Woverflow]


```
x = 1e309; printf("x = 1e309  x = %e\n", x);
```



x = 1e-307   x = 1.000000e-307  
x = 1e-308   x = 1.000000e-308  
x = 1e-309   x = 1.000000e-309  
x = 1e-400   x = 0.000000e+00  
x = 1e-401   x = 0.000000e+00  
x = 1e-402   x = 0.000000e+00  
x = 1e-403   x = 0.000000e+00  
x = 1e-404   x = 0.000000e+00  
x = 1e-405   x = 0.000000e+00  
x = 1e-406   x = 0.000000e+00

x = 1e308   x = 1.000000e+308  
x = 1e309   x = inf


x = 123.45123451234512345   x = 1.23451234512345124e+02


- 
- Non tutti i numeri reali sono rappresentabili
  - Le operazioni aritmetiche, a differenza degli interi, non sono necessariamente esatte
  - Non è un problema per calcoli di piccola taglia
  - Più problematica per problemi di taglia grossa
    - risoluzione di grossi sistemi di equazioni differenziali
    - minimizzazione di funzionali complessi

# Uso di typedef

- Il linguaggio C fornisce il meccanismo del typedef
  - Permette di associare un tipo di dato ad un identificatore
    - `typedef char uppercase;`
    - `typedef int inches;`
    - `typedef unsigned long size_t;`
    - `typedef unsigned char u_char;`



- 
- L'identificatore può essere usato per dichiarare variabili o funzioni, come per i tipi standard
    - uppercase u;
    - u\_char c;
    - inches length, width;

- 
- Si possono abbreviare dichiarazioni lunghe
  - I nomi dei tipi riflettono l'uso delle variabili
  - Gestire differenze fra i sistemi critiche per il programma
    - Es. portare il codice sviluppato per int a 4 byte su un sistema dove sono a 2 byte
    - Sul primo sistema

```
typedef int      tipoint;
```
    - Sul secondo sistema

```
typedef long int tipoint;
```

# Operatore sizeof

- Il C fornisce l'operatore unario sizeof
- Determina il numero di byte necessari per rappresentare un *oggetto*
- Ha la stessa priorità degli altri operatori unari
- sizeof(object) restituisce un intero pari al numero di byte necessari a rappresentare object in memoria
- object può essere
  - un tipo, come int o float
  - un'espressione, come a + b
  - un array o una struttura



```
/* Compute the size of some fundamental types. */
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    printf("The size of some fundamental types is computed.\n\n");
```

```
    printf("    char:%3ld byte \n", sizeof(char));
```

```
    printf("    short:%3ld bytes\n", sizeof(short));
```

```
    printf("    int:%3ld bytes\n", sizeof(int));
```

```
    printf("    long:%3ld bytes\n", sizeof(long));
```

```
    printf("    unsigned:%3ld bytes\n", sizeof(unsigned));
```

```
    printf("    float:%3ld bytes\n", sizeof(float));
```

```
    printf("    double:%3ld bytes\n", sizeof(double));
```

```
    printf("long double:%3ld bytes\n", sizeof(long double));
```

```
    return 0;
```

```
}
```




The size of some fundamental types is computed.


char:	1 byte
short:	2 bytes
int:	4 bytes
long:	8 bytes
unsigned:	4 bytes
float:	4 bytes
double:	8 bytes
long double:	16 bytes

# Usi di `getchar()` e `putchar()`

- Usate per
  - leggere caratteri da standard input
  - scrivere caratteri su standard output
- Solitamente implementate come macro
  - direttive per il preprocessore
  - usate come funzioni
- utilizzano `int`
  - `int` possono rappresentare qualsiasi carattere

- 
- Consideriamo un programma che
    - legge una sequenza di caratteri da standard input
    - li scrive due volte sullo standard output
    - eseguiamo il programma redirezionando file di input su standard input

```
double_out < input.txt
```



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF) {
```

```
        putchar(c);
```


```
        putchar(c);
```

```
    }
```

```
    return 0;
```

```
}
```





```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF) {
```

```
        putchar(c);
```

```
        putchar(c);
```


```
    }
```

```
    return 0;
```

```
}
```

abcdefghijklmno

aabbccddeeffgghhiijjkkllmmnnoo



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF) {
```


```
        putchar(c);
```

```
        putchar(c);
```


```
    }
```

```
    return 0;
```

```
}
```

- 
- EOF: end of file
    - indica la fine del file di input
    - definita in `stdio.h` tipicamente come

```
#define EOF      (-1)
```
    - potrebbe essere definito anche diversamente
    - l'uso della variabile simbolica garantisce la portabilità



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF) {
```


```
        putchar(c);
```


```
        putchar(c);
```

```
    }
```

```
    return 0;
```

```
}
```

- 
- La variabile `c` è dichiarata di tipo `int`
    - EOF può non avere un valore rappresentato da un carattere
      - `char` implementato come `signed char` o `unsigned char`
      - EOF potrebbe essere definito diversamente da `-1`
      - uso di `int` garantisce la portabilità



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    unsigned char c;
```

```
    while ((c = getchar()) != EOF) {
```


```
        putchar(c);
```

```
        putchar(c);
```

```
    }
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    unsigned char c;
```

```
    while ((c = getchar()) != EOF) {
```

```
        putchar(c);
```


```
        putchar(c);
```

```
    }
```

```
    return 0;
```

```
}
```

loop infinito



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF) {
```

```
        putchar(c);
```


```
        putchar(c);
```


```
    }
```


```
    return 0;
```


```
}
```



- 
- `c = getchar()`
    - riceve un valore da standard input
    - lo assegna alla variabile `c`
    - il valore ritornato da questa espressione è `c`
  - `(c = getchar()) != EOF`
    - confronta il valore ritornato dall'espressione interna con `EOF`
    - `TRUE` se i due valori sono diversi

- 
- Tutte le parentesi sono necessarie
  - L'espressione  
    `c = getchar() != EOF`
    - è interpretata, a causa della precedenza degli operatori, come  
    `c = (getchar() != EOF)`  
    che ha una semantica diversa
  - `putchar(c)`, stampa su standard output il carattere identificato da `c`

- 
- I caratteri sono rappresentati anche come numeri interi
  - I caratteri sono ordinati come gli interi
    - ‘a’ + 1 ha valore ‘b’
    - ‘b’ + 1 ha valore ‘c’
    - ‘z’ – ‘a’ ha valore 25
    - ‘A’ > ‘a’
    - ‘A’ – ‘a’ ha lo stesso valore di ‘B’ - ‘b’



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF)
```

```
        if (c >= 'a' && c <= 'z')
```


```
            putchar(c + 'A' - 'a');
```

```
        else
```

```
            putchar(c);
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c;
```

```
    while ((c = getchar()) != EOF)
```

```
        if (c >= 'a' && c <= 'z')
```

```
            putchar(c + 'A' - 'a');
```

```
        else
```

```
            putchar(c);
```

```
    return 0;
```

```
}
```

AbcDE6ygTr56


ABCDE6YGTR56




Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
1	01	Start of heading	33	21	!	65	41	A	97	61	a	129	81	ù	161	A1	í	193	C1	ł	225	E1	β
2	02	Start of text	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	ṽ	226	E2	Γ
3	03	End of text	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	ṭ	227	E3	π
4	04	End of transmit	36	24	\$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
5	05	Enquiry	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
6	06	Acknowledge	38	26	&	70	46	F	102	66	f	134	86	ã	166	A6	ª	198	C6	‡	230	E6	μ
7	07	Audible bell	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	‖	231	E7	τ
8	08	Backspace	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	ℓ	232	E8	Φ
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	¬	201	C9	ℝ	233	E9	Θ
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	ℓ	234	EA	Ω
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	℥	235	EB	Θ
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¾	204	CC	℥	236	EC	∞
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	=	237	ED	∞
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n	142	8E	Ë	174	AE	«	206	CE	‡	238	EE	ε
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o	143	8F	Ä	175	AF	»	207	CF	ℓ	239	EF	Π
16	10	Data link escape	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	☐	208	D0	ℓ	240	FO	≡
17	11	Device control 1	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	☐	209	D1	℥	241	F1	±
18	12	Device control 2	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	☐	210	D2	π	242	F2	≥
19	13	Device control 3	51	33	3	83	53	S	115	73	s	147	93	ó	179	B3		211	D3	ℓ	243	F3	≤
20	14	Device control 4	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	†	212	D4	ℓ	244	F4	[
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	‡	213	D5	℥	245	F5	]
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	‖	214	D6	π	246	F6	÷
23	17	End trans. block	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	π	215	D7	‡	247	F7	≈
24	18	Cancel	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	¶	216	D8	‡	248	F8	•
25	19	End of medium	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	‖	217	D9	¶	249	F9	•
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	‖	218	DA	¶	250	FA	•
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{	155	9B	÷	187	BB	¶	219	DB	■	251	FB	√
28	1C	File separator	60	3C	<	92	5C	\	124	7C		156	9C	£	188	BC	¶	220	DC	■	252	FC	Δ
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	¶	221	DD	■	253	FD	z
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~	158	9E	℔	190	BE	¶	222	DE	■	254	FE	■
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□	159	9F	ƒ	191	BF	¶	223	DF	■	255	FF	□

# Funzioni matematiche

- Il linguaggio C non contiene funzioni matematiche predefinite
- Funzioni matematiche sono disponibili nella libreria matematica
- Fa parte della libreria standard
- Esempi
  - `sqrt()`
  - `pow()`
  - `log()`
  - `sin()`
  - `cos()`

- 
- In genere
    - ricevono parametri di tipo double
    - ritornano un risultato di tipo double
  - Vediamo un esempio con l'uso di
    - `sqrt()`: un parametro
    - `pow()`: due parametri






```
#include <math.h>
#include <stdio.h>
```

```
int main(void)
{
    double x;

    printf("\n%s\n%s\n%s\n\n",
        "The square root of x and x raised",
        "to the x power will be computed.",
        "---");
    while (1) { /* do it forever */
        printf("Input x: ");
        scanf("%lf", &x);
        if (x >= 0.0)
            printf("\n%15s%22.15e\n%15s%22.15e\n%15s%22.15e\n\n",
                "x = ", x,
                "sqrt(x) = ", sqrt(x),
                "pow(x, x) = ", pow(x, x));
        else
            printf("\nSorry, your number must be nonnegative.\n\n");
    }
    return 0;
}
```



The square root of x and x raised  
to the x power will be computed.

---

Input x: 2

```
      x = 2.0000000000000000e+000
    sqrt(x) = 1.414213562373095e+000
  pow(x, x) = 4.000000000000000e+000
```

Input x:

# Valore assoluto

- La libreria standard include due funzioni per calcolare il valore assoluto
  - `abs()`
    - valore assoluto di parametri `int`
    - prototipo definito in `stdlib.h`
  - `fabs()`
    - valore assoluto di parametri `double`
    - prototipo definito in `math.h`

# Conversioni e cast

- Una espressione aritmetica come  $x + y$  ha
  - un valore
  - un tipo
- Se  $x$  e  $y$  sono di tipo `int` allora  $x + y$  è di tipo `int`
- Se  $x$  e  $y$  sono di tipo `short` allora  $x + y$  è di tipo `int`
  - Nelle espressioni i valori `short` vengono convertiti a `int`

# Promozioni a intero

- Un char o uno short, sia signed sia unsigned, si può usare in ogni espressione dove può essere usato un int o un unsigned int
- Se tutti i valori del tipo originale possono essere rappresentati da un int
  - il valore è convertito in int
  - altrimenti è convertito in unsigned int


- 
- Ad esempio

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


- La variabile `c` di tipo `char` è un parametro della `printf()`
- Il parametro è effettivamente una espressione
- Il tipo della espressione, a causa della promozione a intero, è di tipo `int`


# Conversioni aritmetiche

- Durante la valutazione degli operandi di un operatore binario possono avvenire delle conversioni
- Se ad esempio
  - `int i;`
  - `float f;`
  - `i + f;`
  - `i` viene promosso a `float`

- 
- Se uno dei due operandi è di tipo long double, l'altro operando viene convertito in long double
  - Altrimenti, se uno dei due operandi è double, l'altro operando viene convertito in double
  - Altrimenti, se uno dei due operandi è di tipo float, l'altro operando viene convertito in float
  - Altrimenti viene applicata la conversione a intero su entrambi gli operandi



- 
- Nell'ultimo caso vengono applicate le regole
    - se uno dei due operandi è unsigned long, l'altro operando viene convertito in unsigned long
    - altrimenti, se uno dei due operandi è long e l'altro unsigned
      - se il tipo long può rappresentare tutti i valori di un unsigned, l'unsigned è convertito in long
      - altrimenti, entrambi vengono convertiti in unsigned long

- 
- Altrimenti, se uno dei due è long, l'altro viene convertito in long
  - Altrimenti se uno è unsigned, l'altro viene convertito in unsigned
  - Altrimenti, entrambi gli operandi sono di tipo int

## Dichiarazioni

char c;  
long l;  
float f;

short s;  
unsigned u;  
double d;

int i;  
unsigned long ul;  
long double ld;

**Espressione**

**Tipo**

**Espressione**

**Tipo**

$c - s / i$

$u * 7 - i$

$u * 2.0 + 1$

$f * 7 - i$

$c + 3$

$7 * s * ul$

$c + 5.0$

$ld + c$

$d + s$

$u - ul$

$2 * i / l$

$u - l$

## Dichiarazioni

char c;  
long l;  
float f;

short s;  
unsigned u;  
double d;

int i;  
unsigned long ul;  
long double ld;

### Espressione

### Tipo

### Espressione

### Tipo

$c - s / i$

int

$u * 7 - i$

unsigned

$u * 2.0 + 1$

double

$f * 7 - i$

float

$c + 3$

int

$7 * s * ul$

unsigned long

$c + 5.0$

double

$ld + c$

long double

$d + s$

double

$u - ul$

unsigned long

$2 * i / l$

long

$u - l$

dipende dal sistem

# Cast

- Quelle viste sono conversioni implicite
- Esistono anche conversioni esplicite
  - Chiamate cast

```
int i;  
double f;  
f = (double) i;
```
  - La variabile `i` non viene modificata
  - Viene modificato il tipo ritornato dall'espressione `i`