Elementi C

```
int main() {
    int a,b;
    a = 2;
    b = 3;
    int c = 32;
}
```

spazio indentazione

Tipo	Variabile	Bit
char	carattere	8
short	intero	16
int	intero	32
long	intero	32
float	razionale	32
long long int	intero	64
double	razionale	64
long double	razionale	96

Caratteri speciali: non nel nome variabile

```
. , ; : + - = / \* < > ! ? ~ @ # $ % ^ & | " ` '
```

```
int main() {
    int a; a = -32; int b = a;
}
```

```
int main(){ int a = 32; int b; }
```

nei caratteri alfanumerici per nome variabile
potete usare A_b

Operazione	Simbolo	Esempio
somma	+	a + b
differenza	-	a - b
prodotto	*	a * b
quoziente	/	a / b

```
int main() {
    int a = 3;
    int b = 4;
    int c = a / b;
    printf("%d", c);
```

```
printf("%d, %d, %d", a, b, c);
```

una sola stringa di formattazione con 3 descrittori %d per intero

3 variabili di tipo int

```
gcc -o app.exe prog.c
./app.exe
```

output

03,4,0

Descrittori per output

Descrittore	Tipo
%f, %e, %g	float, double
%d, %i	int
%c	char (singolo)
%s	char (stringa)

```
float d;
d = a/b;
printf("a: %d, b = %d, d = %f\n", a, b, d);
```

\n vai a capo; newline

output

a: 3, b = 4, d = 0.000000

perché 0?

d = a / b

① a/b : operazione / tra interi ∅

② assegno d = ∅

```
float a, b;
a = 3;
b = 4;
float c = a/b;
```

a/b: operaz. tra float

equivale a 3.000/4.000

c = 0.75

d = 3/4 → 0

d = 3./4 → 0.75

float → int

d = 3/4. → 0.75

d = 3./4. → 0.75

float / int → promosso a float dal compilatore

```
int a=3;
int b=4;
float c = (float a)/b;
```

casting: tratta a come float

equivale a 3./4

```
c = b/a;
```

int c = a/b divisione

int r = a%b; resto della divisione

c [0] r [3]

```
int a = 3;
float b = 4;
float c = a/b;
float d = 1/(a*b);
d = 1/a*b;
```

1) $\frac{1}{a \times b}$

2) $\frac{1}{a} \times b$

```
int a = 3;
int b = 4;
float c = 1./a*b;
```

evita di avere op. tra interi

```
++ (post) -- (post)
++ (pre) -- (pre) ! -(unario)
* / %
+ - (binario)
< <= > >=
== !=
&&
||
= += -= *= /=
```

a + b   +: operatore binario

(...) + (...)

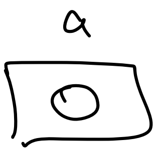
LHS: Left hand side   RHS: right hand side

-a : - operatore unario

-3

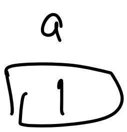
int a=0.

a = a+1



① a+1 = 1

② a = 1



a (=) a+1 — operatore

LHS   RHS

a++; equivale a a = a+1

++a;

int a=0;

int b = a++;   1) assegna a → b 2) incrementa a.   b=0; a=1

int b = ++a;   b=1, a=1   1) incrementa a 2) assegna a → b

a++;   (a = 2)

a++;   a → 3

a += 3; equivale a a = a+3

-=

*=

a /= b

a = a/b

d = a+1

d = 1+a

a non è modificato.
d viene modificato.

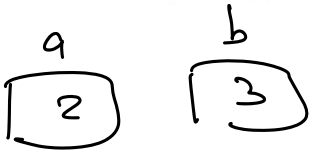
a++;
a--;

a viene modificato.

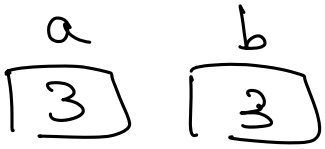
a+1 = a NO

2 = a; NO

++ (post) -- (post)
++ (pre) -- (pre) ! -(unario)
* / %
+ - (binario)
< <= > >=
== !=
&&
||
= += -= *= /=



a = b



assegnazione

int a=2, b=3;
int c;

c = a==b;

assegn. confronto

int a=2;
int b=3;

a = b assegnazione

a == b confronto uguaglianza.

a == b

a > b

a < b

a <= b

a >= b

confront.

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

output

c: 0 False

c = a <= b;

c = (a<=b);

c = 1 Vero.

1: Vero 0: Falso

A	B	AND	OR	NOT
1	1	1	1	
1	0	0	1	
0	1	0	1	
0	0	0	0	

&& AND

|| OR

A AND B

int c, d;

c = a <= b;

d = a > b;

A	NOT A
1	0
0	1

!A

int e = a && b;

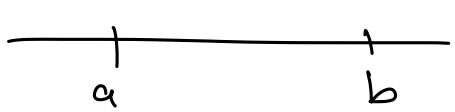
AND

int f = (a <= 2.3) && (b >= -27);

int g = (a < b) || (b >= 3.2);

OR

int g = ! (a < b) equivale a a >= b



c ∈ [a, b]

$a <= c <= b$ Non in C.

$c \in [a,b]$ $c \geq a$ AND $c \leq b$

$(a <= c) \&\& (c <= b)$

$c \in (0, b]$ $(c > 0) \&\& (c <= b)$

$(0 < c) \&\& (c <= b)$

$c > 0$

$0 < c$

c

3

$0 = c$

```
int main() {
    int c;
    c = 3;
    printf("c = %d\n", c);
}
```

⟶ output

Input con scanf(--)

```
int main() {
    int c;
    printf("Inserisci valore di c \n");
    printf("c = ");
    scanf("%d", &c);
}
```

& e commerciale

descrittore

&variabile

```
float x;
printf("inserisci valore razionale x: ");
scanf("%f", &x);
printf("x = %f\n", x);
```

./app.exe esegue

output

inserisci valore razionale x: pippo IN VIO

-3.2

x = -3.2

Descrittore	Tipo
%f	float
%lf	double
%Lf, %llf	long double
%d, %i	int
%u	unsigned int
%Lu	unsigned long long int
%c	char (singolo)
%s	char (stringa)

standard input/output

header file della lib. matem.

prog.c

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

int main() {
    float x = 2.3;
    float y = sqrt( x );   radice quadrata
    printf("sqrt(%f) = %f\n", x, y );
```

```
gcc -o app.exe prog.c -lm
./app.exe
sqrt(2.3) = 1.4723137
```

Da math.h

```
/* The above constants are not adequate for computation using `long double's.
   Therefore we provide as an extension constants with similar names as a
   GNU extension.  Provide enough digits for the 128-bit IEEE quad.  */
#ifdef __USE_GNU
# define M_El          2.718281828459045235360287471352662498L /* e */
# define M_LOG2El      1.442695040888963407359924681001892137L /* log_2 e */
# define M_LOG10El     0.434294481903251827651128918916605082L /* log_10 e */
# define M_LN2l        0.693147180559945309417232121458176568L /* log_e 2 */
# define M_LN10l       2.302585092994045684017991454684364208L /* log_e 10 */
# define M_PIl         3.141592653589793238462643383279502884L /* pi */
# define M_PI_2l       1.570796326794896619231321691639751442L /* pi/2 */
# define M_PI_4l       0.785398163397448309615660845819875721L /* pi/4 */
# define M_1_PIl       0.318309886183790671537767526745028724L /* 1/pi */
# define M_2_PIl       0.636619772367581343075535053490057448L /* 2/pi */
# define M_2_SQRTPIl   1.128379167095512573896158903121545172L /* 2/sqrt(pi) */
# define M_SQRT2l      1.414213562373095048801688724209698079L /* sqrt(2) */
# define M_SQRT1_2l    0.707106781186547524400844362104849039L /* 1/sqrt(2) */
#endif
```

```
#include <math.h>
int main() {
    float xin = 35; /* xin angolo in gradi */
    float xout;
    xout = (xin / 180.) * M_PI;
    float y = sin(xout);
    printf("sin(%f gradi) = %f\n", xin, y);
```

Case Sensitive

```
int xin = 35.
float xout = (xin / 180) * M_PI;
sin(xout) ≡ ∅

  float xin;
printf("angolo in gradi:");
scanf("%f", &xin);
```

```
c:\app.exe
angolo in gradi: -7 [INVIO]
```