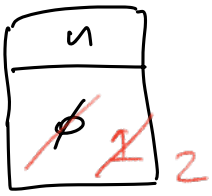
Operazione	Simbolo	Esempio
somma	+	a + b
differenza	-	a - b
prodotto	*	a * b
quoziente	/	a / b

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

```
int n = 0;
n = n+1;
n = n+1;
```



n += 1; equivalent ⟷ n = n + 1;

n += 2; ⟷ n = n + 2;

n *= 2; ⟷ n = n * 2

n++; ⟷ n = n + 1;

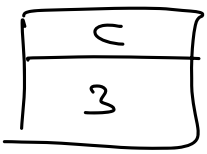
n = 0;

```
n++;
n += 1
n = n+1
```

lo stesso effetto su n.

priorità

```
int a = 1, b = 2;
int c = a*b + a;
int d = a*b - a;
  = -2 + b*a;
```



```c
#include <stdio.h>

int main() {
  double a = 4, b = 3., c=2, d;


  printf("a = %f \t b = %lf \t c = %lf\n\n", a,b,c);

  d = a*b/c;
  printf("a*b/c = %f\n\n", d);


  d = a/b*c;
  printf("a/b*c = %f\n\n", d);


  d = a/(b*c);
  printf("a/(b*c) = %f\n\n", d);


    return 0;
}
```

\ backslash

/ slash.

\n: vai a capo

\t: tab

4/6 = 0.666- -

```
shamacmini:material rahatlou$ gcc -o /tmp/app priorita.c
shamacmini:material rahatlou$ /tmp/app
a = 4.000000     b = 3.000000     c = 2.000000

a*b/c = 6.000000

a/b*c = 2.666667

a/(b*c) = 0.666667

shamacmini:material rahatlou$
```

$\longleftrightarrow$ (a*b)/c

$\longleftrightarrow$ (a/b)*c

(a)/(b)

(1)/((b)*(c))

|||

1/(b*c)

Descrittori

numeri razionali in output %f

%N.Mf $\longrightarrow$ M cifre decimali

$\searrow$ almeno N cifre (incluso .)

a = 2,
printf("%5.3f", a)

2.000

double w = 0.0123456789

pnutf("%.5f", w)

0.01235

~/Sites/Didattica/2025.labcalc/material — rahatlou@labcalc2: ~ — vim descrittore.c

```c
#include <stdio.h>

int main() {
    double a = 12345678.123456789;
    double b = 0.0001234567;

    printf("Con %%f:senza specificare numero di decimali\n");
    printf("Valore di a = %f\n", a);
    printf("Valore di b = %f\n\n", b);

    printf("Con %%5.3f: almeno 5 cifre (incluso .) e 3 decimali\n");
    printf("a = %5.3f\n", a);
    printf("b = %5.3f\n\n", b);

    printf("Con %%5.3e: notazione esponenziale/scientifica con 3 decimali\n");
    printf("a = %5.3e\n", a);
    printf("b = %5.3e\n\n", b);

    printf("Con %%5.3g: notazione compatta con 3 cifre significative ed almeno 5 cifre totali (inclu
so .) \n");
    printf("a = %5.3g\n", a);
    printf("b = %5.3g\n\n", b);

    printf("Con %%.15f: con 15 cifre decimali\n");
    printf("a = %.15f\n", a);
    printf("b = %.15f\n\n", b);


    return 0;
}
```

```
shamacmini:material rahatlou$ gcc -o /tmp/app descrittore.c
shamacmini:material rahatlou$ /tmp/app
Con %f:senza specificare numero di decimali
Valore di a = 12345678.123457
Valore di b = 0.000123

Con %5.3f: almeno 5 cifre (incluso .) e 3 decimali
a = 12345678.123
b = 0.000

Con %5.3e: notazione esponenziale/scientifica con 3 decimali
a = 1.235e+07
b = 1.235e-04

Con %5.3g: notazione compatta con 3 cifre significative ed almeno 5 cifre totali (incluso .)
a = 1.23e+07
b = 0.000123

Con %.15f: con 15 cifre decimali
a = 12345678.123456789180636
b = 0.000123456700000

shamacmini:material rahatlou$
```

```
#include <math.h>
#include <stdio.h>
int main() {
    printf("pi = %f\n", M_PI);
```

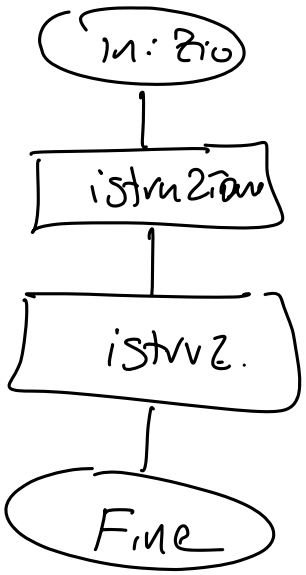
pi = 3.141590

%.15f

/usr/include/math.h

```
/* Some useful constants.  */
#if defined __USE_MISC || defined __USE_XOPEN
# define M_E            2.7182818284590452354   /* e */
# define M_LOG2E        1.4426950408889634074   /* log_2 e */
# define M_LOG10E       0.43429448190325182765  /* log_10 e */
# define M_LN2          0.69314718055994530942  /* log_e 2 */
# define M_LN10         2.30258509299404568402  /* log_e 10 */
# define M_PI           3.14159265358979323846  /* pi */
# define M_PI_2         1.57079632679489661923  /* pi/2 */
# define M_PI_4         0.78539816339744830962  /* pi/4 */
# define M_1_PI         0.31830988618379067154  /* 1/pi */
# define M_2_PI         0.63661977236758134308  /* 2/pi */
# define M_2_SQRTPI     1.12837916709551257390  /* 2/sqrt(pi) */
# define M_SQRT2        1.41421356237309504880  /* sqrt(2) */
# define M_SQRT1_2      0.70710678118654752440  /* 1/sqrt(2) */
#endif
```

Diagramma di flusso



messaggio all'utente

prendi valore da utente

Fai calcolo

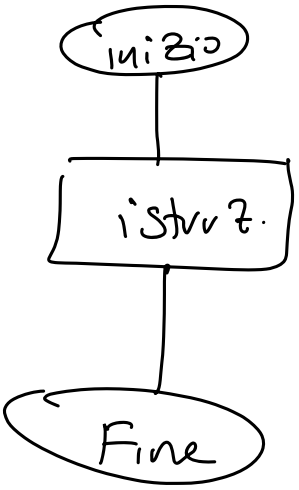
stampa calcolo.

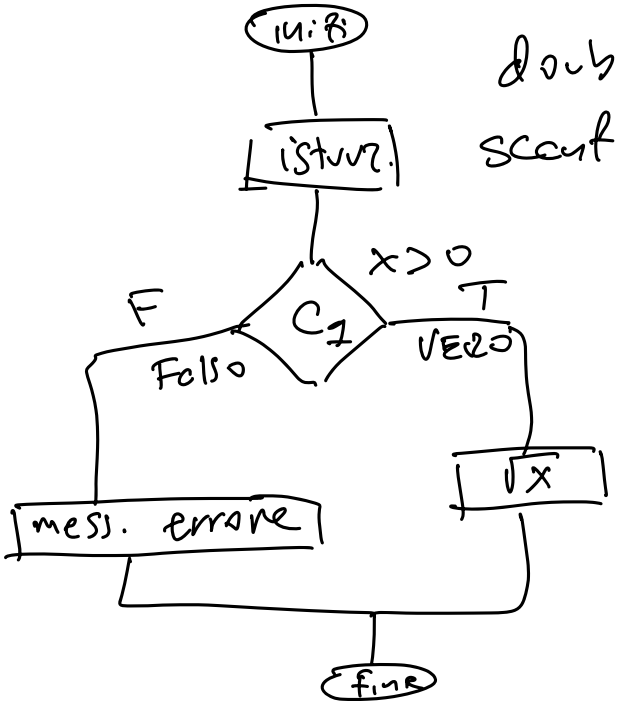
Flusso lineare senza decisioni:

Teorema di Böhm – Jacopini:

algoritmo sempre composto da – istruzione
– decisione
– iterazione

Flusso lineare





double x;
scanf("%lf", &x);

```
#include <math.h>
#include <stdio.h>
int main() {
    double x;
    printf("inserisci x>0: ");
    scanf("%lf", &x);
    if (x>0) {          ⟶ x >=
        printf("sqrt(%lf) = %lf\n", x, sqrt(x));
    } else {
        printf("%lf < 0 !!\n", x);
    }
}
```

Costrutto if

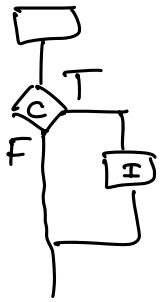
cond.?

```
if (x>0) printf("x positivo");

if (x>0) {
    printf("x positivo");
}
```

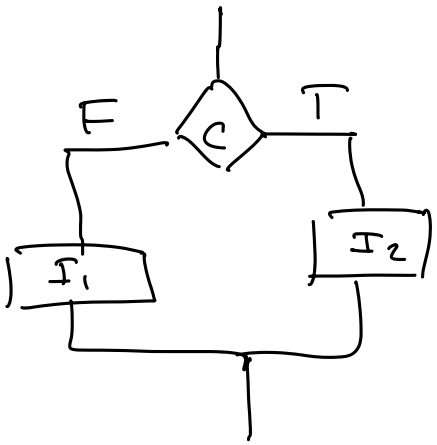
```
if (x>0) {
    printf("x giusto");
    double y = sqrt(x);
    printf("sqrt(%f) = %f\n", x, y);
}
```

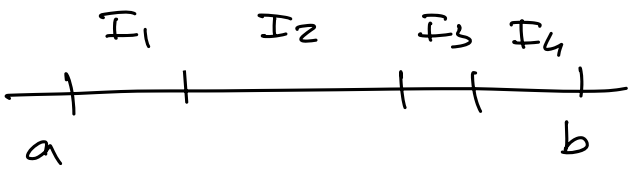
```
int main() {

    ==========

    if (--) {
        if(--) {--
            ==
        }
    }

    if (--) {
        ==========
    }

}
```

```
if ( condiz) {
    ≡≡≡
}
```




```
if( condizione ) {
    ≡≡≡
} else {
    ≡≡≡
}
```





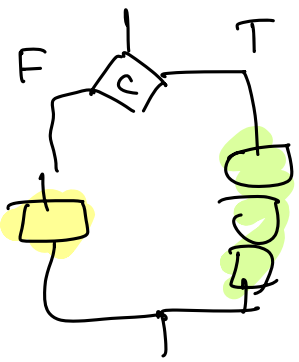
Come richiede $x \in [a,b]$

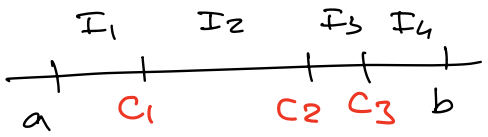
```
double x;
scanf("%lf", &x)
if ( a<x<b ) {
```

$a \in [a,b] \equiv a<x$ AND $x<b$

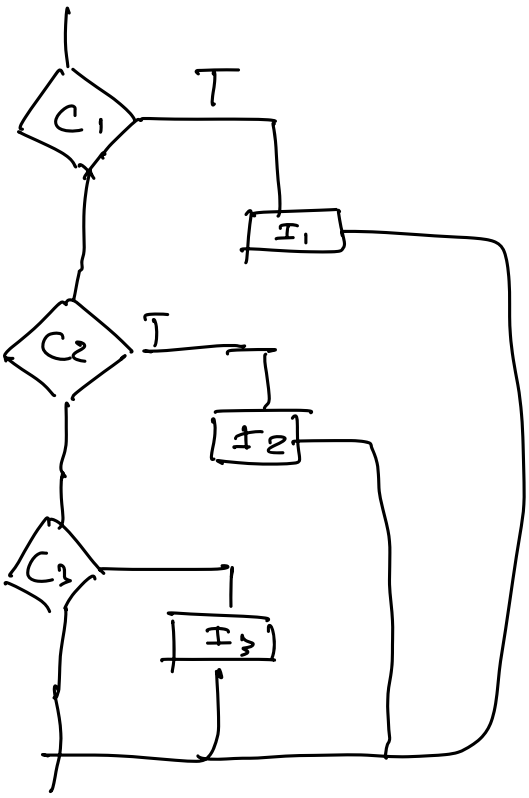
&& AND LOGICO

```
if ( x>=a && x<=b ) {
    ≡≡≡
} else {
    printf("x non valido \n");
}
```





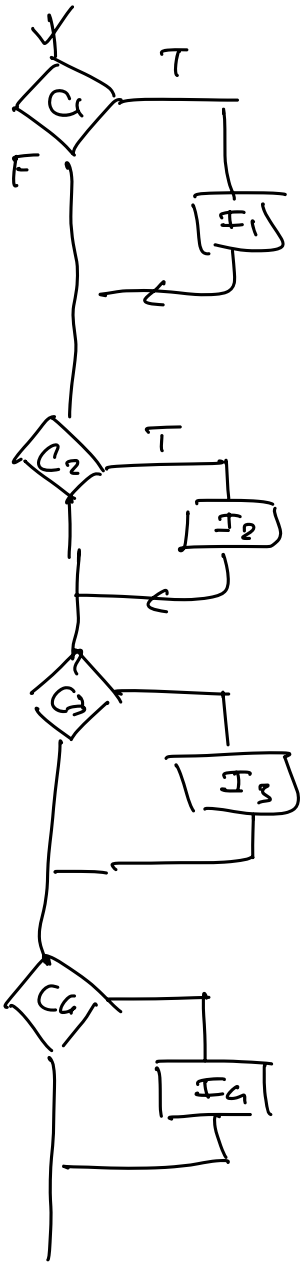

```
if ( x>=a && x<=b ) {
    if ( x>a && x<c1 ) { // in I1
        ≡
    } else if ( x>c1 && x<c2 ) { // in I2
        ≡
    } else if ( x>c2 && x<=c3 ) { // in I3
        ≡
    } else { // sono in I4
        ≡
    }
}
```



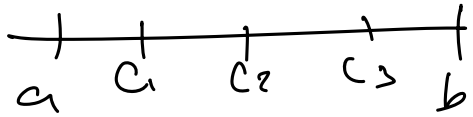
if / else if / else

esclusivi

```
if( x>=a && x < c1)?
    ==
{
if ( x >=c1 && x <= c2) ?
    ==
{
if ( x >= c2 && x <= c3) ?
    ==
{
if ( x >= c3 && x<=b ) ?
    ==
{
```



if non esclusivi:



C

$x \notin [a,b]$ $\quad\quad$ $x < a \;||\; x > b$

$x \in [a,b]$ $\quad = \quad$ $x >= a \;\&\&\; x <= b$

&& AND logico

|| OR logico.

!(A AND B) !A OR !B

!(x >= a && x <= b) negazione!

equivalente a

x < a || x > b

!(x > a) x <= a

Conversione di angoli

```
printf("inserisci angolo");
double x;
scanf("%lf", &x);
int scelta = 0;
printf("Indicare gradi (0) oppure rad (1):");
scanf("%d", &scelta);
if (scelta == 0) {
    printf("in radianti: %.1f \n", (x/180.) * M_PI);
} else if (scelta == 1) {
    printf("in gradi: %.1f \n", (x(M_PI) * 180.);
} else {
    printf("scelta non valida %d \n", scelta);
}
```

```
if (x >= 0) {
```

≡

```
} else if (x < 0) {
```

≡

```
}
```

else

problema acquisizione input:

$x \in [a, b]$

alternativa ovvero solo questo

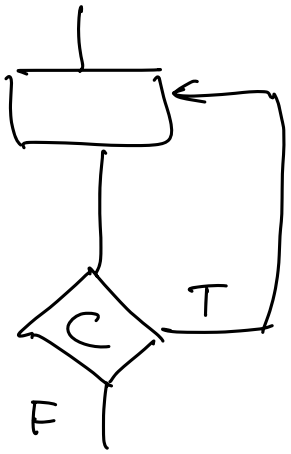
⇒ iterazioni:

voglio: x >= a && x <= b

```
double x;
double a = 1, b = 2;
do {
    printf("inserisci a in [1,2]: ");
    scanf("%lf", &x);
} while (x < a || x > b);
```

negazione della cond. voluta

ciclo do/while



```
int nstupido = 0;

do {
    printf("inserisci a in [1,2]: ");
    scanf("%lf", &x);
    int cond = x < a || x > b;
    if (cond) {
        printf("Hai sbagliato valore! Riprova!\n");
        nstupido++;
    }
} while (cond);

if (nstupido > 0) {
    printf("complimenti! hai sbagliato %d volte\n", n);
}
```