

# Tecniche della Programmazione, lez. 3

- Introduzione allo sviluppo ed esecuzione di programmi

# Sintassi ...

Un programma è una sequenza di istruzioni, scritte nel linguaggio di programmazione in uso (Per noi il C).

E' un lungo paragrafo, costituito da tante frasi, una dopo l'altra (le istruzioni).

Per scrivere bene una frase bisogna seguire delle regole, questo vale per qualsiasi linguaggio ...

La **SINTASSI** è la grammatica da seguire per scrivere le frasi (le istruzioni)

$\text{deltaQuadro} = b*b - 4*a*c;$

**SI**

Segue le regole grammaticali

Delta Quadro = bb - ; 4a\*c

**NO**

Non segue le regole grammaticali

## ... e Semantica

Una frase scritta grammaticalmente bene può avere un significato, oppure no:

Lucilla mangia la mela VS. La mela mangia Lucilla

La **SEMANTICA** è il significato dell'istruzione ... e anche il significato deve essere giusto, per avere un programma corretto

```
stampaIntero(47);
```

SI (probabilmente)

```
stampaIntero(4a7.6);
```

NO

```
sqrt(47.1);
```

;) )

```
sqrt(marco)
```

probabilmente NO

# Memento

Quando un programma viene eseguito,  
viene eseguita la sequenza delle sue istruzioni,  
una istruzione alla volta ... secondo la sequenza

finché il programma finisce

oppure va in crash

Di solito noi scriviamo da sinistra a destra e  
dall'alto verso il basso ...

e l'esecutore delle istruzioni si adegua: quello è  
l'ordine di esecuzione delle istruzioni

# MEMENTO (gia' visto ...)

-ANALISI: QUALI dati? FATTI come (struttura)?

- INPUT numeri reali (double)
- OUTPUT numero reale
- dati per calcoli intermedi ...
- idea:  $\text{prod} \leftarrow b \cdot h$   
 $\text{area} \leftarrow \text{prod} / 2$

SINTESI ... =

- 1) prendere da INPUT (**LETTURA**) i valori da associare a b e h
- 2) calcolare prod (**ASSEGNAZIONE**)
- 3) calcolare area (**ASSEGNAZIONE**)
- 4) fornire in OUTPUT (**SCRITTURA**) il valore di area

C:\Users\marco\Desktop\MARCO\MARCO\c...

```
12
5
il valore dell'area di un triangolo avente base = 12 e
altezza = 5 e' 30
-----
Process exited after 6.051 seconds with return value 0
Press any key to continue . . .
```

# Sviluppo di un programma

PROGRAMMAZIONE ... =

```
#include<stdio.h>

int main() {

    double b, h;

    double area;

    double prod;

    scanf("%lf %lf", &b, &h);

    prod = b*h;

    area = prod/2;

    printf("il valore dell'area di un
    triangolo avente base = %g e
    altezza = %g e` %g\n", b, h,
    area);
```

n 0;

IT

programma  
eseguibile  
ottenuto a partire  
dal programma C

OUTPUT

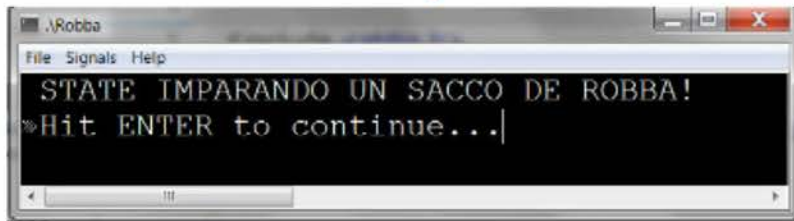
# ISTRUZIONE DI STAMPA

È POSSIBILE SCRIVERE UN'ISTRUZIONE DI STAMPA!

⇒ LA SEMANTICA DI UN'ISTRUZIONE DI STAMPA È CHE VIENE STAMPATO "QUALCOSA" (UNA FRASE, UN VALORE ...) SULLO SCHERMO DELL'UTENTE

⇒ LA SINTASSI È `printf (    );`   
 QUA DENTRO CI VA QUELLO CHE VUOI STAMPARE FRA DOPPI APICI.

`printf("STATE IMPARANDO UN SACCO DE ROBBA!");` QUELLO CHE SCRIVE IL PROGRAMMATTORE



QUELLO CHE VEDE L'UTENTE CHE ESEGUE IL PROGRAMMA

# Istruzioni, funzioni e librerie

L'"istruzione di stampa" discussa prima consiste in realtà in una chiamata alla funzione `printf`.



"Chiamare una funzione" significa "richiedere l'esecuzione della funzione" ...

`printf()` è una funzione non direttamente esistente nel linguaggio C, ma disponibile in un **modulo di programma (libreria)**.

Per cui nel programma, all'inizio, dobbiamo indicare la libreria della quale ci serviremo (le cui funzioni chiameremo nel programma), attraverso una **direttiva include**

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

# Istruzioni, funzioni e librerie

L' "istruzione di stampa" discussa prima consiste in realtà in una chiamata alla funzione `printf`.



"Chiamare una funzione" significa "richiedere l'esecuzione della funzione" ...

`printf()` è una funzione non direttamente esistente nel linguaggio C. Per usarla bisogna indicare nel programma dove la sua definizione e il relativo codice eseguibile possono essere trovati.

Queste definizioni sono disponibili tramite la "libreria di input/output" `stdio.h`

Esistono molte librerie, contenenti funzioni definite per gli scopi più diversi, ed utili nella costruzione dei programmi.

Una libreria ("library") è il **modulo software** (parte di un programma, predefinita e disponibile nell'ambiente di programmazione, che usiamo per fare i programmi).

È una collezione di programmi già fatti che possiamo usare nei nostri programmi ... possiamo anche scrivere una libreria per conto nostro, e poi usarla in un programma

Per cui nel programma, all'inizio, dobbiamo indicare la libreria della quale ci serviremo (le cui funzioni chiameremo nel programma), attraverso una **direttiva include**

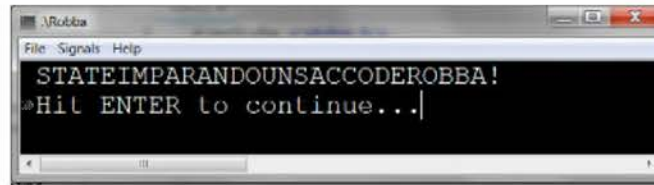
```
#include <stdio.h>
```



# SPAZI E RITORNI A CAPO

LO SPAZIO È UN CARATTERE COME TUTTI GLI ALTRI E COME TALE VA TRATTATO!

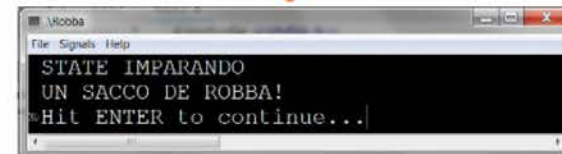
```
printf("STATE");  
printf("IMPARANDO");  
printf("UN");  
printf("SACCO");  
printf("DE");  
printf("ROBBA!");
```



LE ISTRUZIONI `printf` SUCCESSIVE ALLA PRIMA SCRIVONO A PARTIRE DALLA POSIZIONE NELL'OUTPUT IMMEDIATAMENTE SUCCESSIVA A QUELLA DELL'ULTIMO CARATTERE SCRITTO DALL'ISTRUZIONE `printf` PRECEDENTE.

```
printf("STATE IMPARANDO \nUN SACCO DE ROBBA!");
```

PER ANDARE A CAPO, SI UTILIZZA IL CARATTERE NEWLINE `\n`



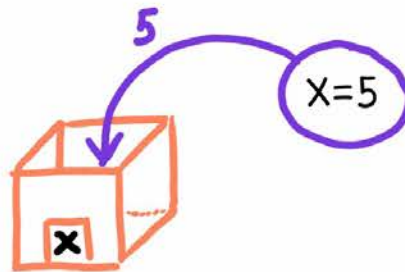
# VARIABILE

IN MATEMATICA, UNA VARIABILE È UN CARATTERE ALFABETICO CHE RAPPRESENTA UN NUMERO ARBITRARIO, SCONOSCIUTO, O NON COMPLETAMENTE SPECIFICATO.

IN INFORMATICA, UNA VARIABILE È UNA PORZIONE DI MEMORIA DESTINATA A CONTENERE DEI DATI CHE POTRANNO ESSERE ACCEDUTI O MODIFICATI DURANTE L'ESECUZIONE DI UN PROGRAMMA.

IN MANIERA PIÙ SEMPLICE:

IN INFORMATICA, UNA VARIABILE È UN CONTENITORE DI VALORI



# Piu' formalmente ...

## Concetto di **VARIABILE**

Una variabile, in un programma, è contemporaneamente

- Un **IDENTIFICATORE** ...
- Una **LOCAZIONE** di memoria  
contraddistinta da un **INDIRIZZO!**
- Un **VALORE**

```
int altezzaMarco;          /* identificatore: altezzaMarco
                           locazione riservata in memoria, con un certo
                           indirizzo (000000000010111010)
                           valore contenuto nella locazione 186          */
```

### Memoria

000000000010111010

000000000000111111

000000000010111010

000000000001000001

**altezzaMarco**

# Piu' formalmente ...

## Concetto di **VARIABILE**

Una variabile, in un programma, è contemporaneamente

- Un **IDENTIFICATORE** cioè il nome della variabile, usato nel programma per ... usarla
- Una **LOCAZIONE** di memoria cioè l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa è contraddistinta da un **INDIRIZZO!**
- Un **VALORE** il valore contenuto nella locazione associata alla variabile

```
int altezzaMarco;
```

Memoria

```
0000000000010111010
```

```
/* identificatore: altezzaMarco  
   locazione riservata in memoria, con un certo  
   indirizzo  
   valore contenuto nella locazione */
```

```
000000000000111111  
000000000000100000  
000000000000100001
```

altezzaMarco

# Piu' formalmente ...

## Concetto di **VARIABILE**

Una variabile, in un programma, è contemporaneamente

- Un **IDENTIFICATORE** cioè il nome della variabile, usato nel programma per ... usarla
- Una **LOCAZIONE** di memoria cioè l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa è contraddistinta da un **INDIRIZZO!**
- Un **VALORE** il valore contenuto nella locazione associata alla variabile

```
int altezzaMarco;
```

```
/* dichiarazione di una variabile denominata altezzaMarco; quando inizierà  
l'esecuzione del programma, verrà riservata in memoria una locazione, capace di  
contenere un intero rappresentato in forma binaria (complemento a 2); questa  
locazione avrà un certo indirizzo. Quando si vuole memorizzare il valore 186 nella  
variabile (assegnazione), si memorizza 186 nella locazione. Quando si vuole usare il  
contenuto della variabile, ad esempio per stamparlo in OUTPUT, si accede alla  
locazione e si usa il valore lì contenuto. */
```

0000000000010111010
---------------------

000000000000111111

000000000000100000

0000000000001000001

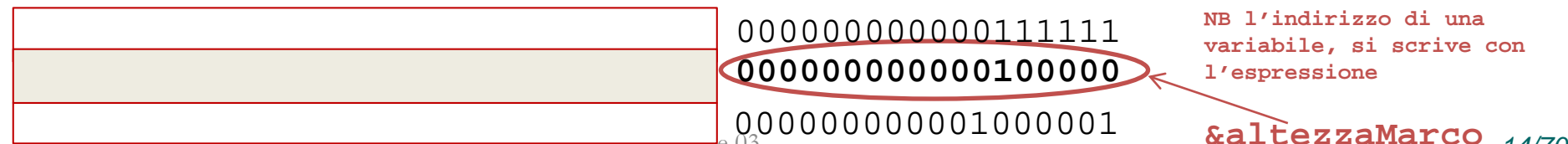
**altezzaMarco**

# Piu' formalmente ...

## Concetto di VARIABILE

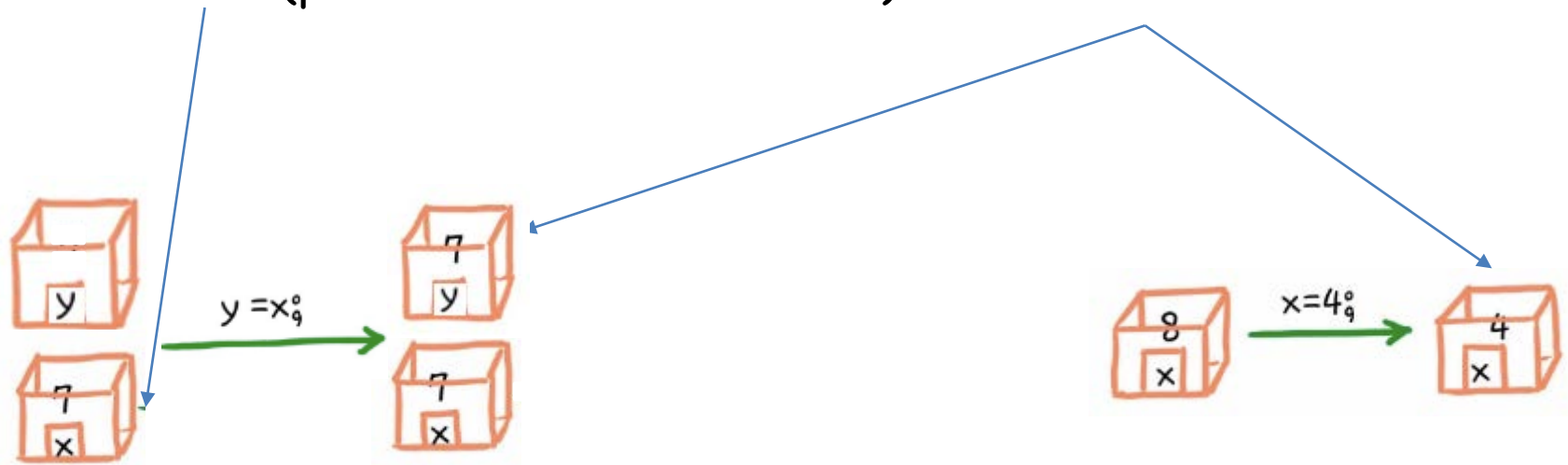
- Una variabile, in programma, è contemporaneamente
- Un **IDENTIFICATORE**      cioè il nome della variabile, usato nel programma per ... usarla)
  - Una **LOCAZIONE** di memoria      cioè l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa è contraddistinta da un **INDIRIZZO!**
  - Un **VALORE**      il valore contenuto nella locazione associata alla variabile

```
int altezzaMarco;      /* dichiarazione di una variabile chiamata  
altezzaMarco; al momento opportuno, verrà riservata in memoria una LOCAZIONE, capace  
di contenere un intero rappresentato in forma binaria (complemento a 2); questa  
locazione avra' un certo INDIRIZZO. Quando si vuole memorizzare il valore 187 nella  
variabile (assegnazione), si memorizza 187 nella locazione. Quando si vuole accedere  
il valore contenuto, ad esempio per stamparlo in OUTPUT, si accede alla locazione. */
```



# E che ci faccio con una VARIABILE?

Visto che una variabile rappresenta nient'altro che una locazione della RAM ... **ci si accede** (per vedere che valore c'è) o **ci si memorizza** un valore ...

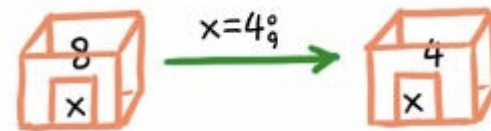


# E che ci faccio con una VARIABILE?

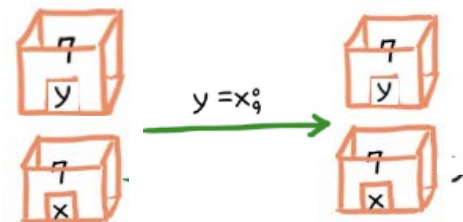
Visto che una variabile rappresenta nient'altro che una locazione della RAM ... **ci si accede** (per vedere che valore c'è) o **ci si memorizza** un valore ...

**NB**

Se si memorizza qualcosa in una variabile (cioè si **ASSEGNA** un **VALORE** ad una **VARIABILE**) il valore precedentemente contenuto nella variabile ... non c'è più (c'è quello che abbiamo assegnato or ora ...)



Se si accede al **VALORE** di una **VARIABILE** (ad esempio per assegnarlo ad un'altra variabile), dopo l'accesso il valore sta ancora nella variabile ... gli accessi non sono «distruttivi»



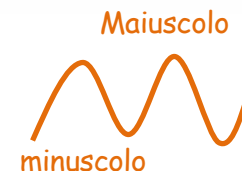


# NOME O IDENTIFICATORE

LE VARIABILI HANNO UN NOME CHE PERMETTE DI IDENTIFICARLE.

- OGNI VARIABILE DEVE AVERE UN NOME DIVERSO
- IL NOME DOVREBBE ESSERE ESPLICATIVO DELLO SCOPO DELLA VARIABILE  
ES: discriminante È UN NOME MIGLIORE DI  $aX^2+bX+c$  PER UNA VARIABILE CHE È DESTINATA A MEMORIZZARE IL DISCRIMINANTE DI UN'EQUAZIONE DI 2° GRADO
- UN NOME È UNA SEQUENZA DI CARATTERI ALFANUMERICI, IL CUI PRIMO CARATTERE DEVE ESSERE ALFABETICO (OPPURE UN UNDERSCORE `_`) E IN GENERE È PREFERIBILMENTE ALFABETICO MINUSCOLO. OGNI PAROLA CHE COSTITUISCE IL NOME, DOPO LA PRIMA, DOVREBBE INIZIARE CON UNA MAIUSCOLA  
ES: radiceReale1, primoNumeroDellaSequenza, minimoCorrente, sequenza3Numeri

camel notation



# MEMENTO (gia' visto ...)

-ANALISI: QUALI dati? FATTI come (struttura)?

- INPUT numeri reali (double)
- OUTPUT numero reale
- dati per calcoli intermedi ...
- idea:  $\text{prod} \leftarrow b \cdot h$   
 $\text{area} \leftarrow \text{prod} / 2$

SINTESI ... =

- 1) prendere da INPUT (**LETTURA**) i valori da associare a b e h
- 2) calcolare prod (**ASSEGNAZIONE**)
- 3) calcolare area (**ASSEGNAZIONE**)
- 4) fornire in OUTPUT (**SCRITTURA**) il valore di area

# Sviluppo di un programma

PROGRAMMAZIONE ... =

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

```
int main() {  
    double b, h;  
    double area;  
    double prod;
```

le variabili  
hanno un tipo

```
scanf("%lf %lf", &b, &h);
```

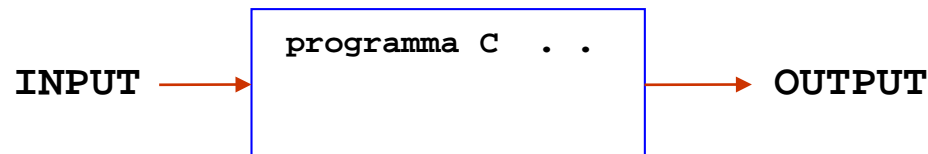
```
prod = b*h;
```

```
area = prod/2;
```

```
printf("il valore dell'area di un  
triangolo avente base = %g e  
altezza = %g e' %g\n", b, h,  
area);
```

```
return 0;
```

```
}
```



**Tipo** LE VARIABILI HANNO UN TIPO CHE RAPPRESENTA L'INSIEME DEI VALORI CHE LA VARIABILE PUÒ ASSUMERE E L'INSIEME DI OPERAZIONI PERMESSE SU QUEI VALORI.

## Alcuni tipi

- int** un sottoinsieme dei numeri interi (tutti quelli rappresentabili in forma binaria, in complemento a 2, in una certa quantità di memoria (stabilita di solito dall'ambiente di programmazione ... di solito 32 bit ...))
- float** un sottoinsieme dei numeri reali (tutti quelli rappresentabili in forma binaria, in Floating Point, in una certa quantità di memoria ... es. non meno di 32 bit ...)
- double** un sottoinsieme dei numeri reali (tutti quelli rappresentabili in forma binaria, in Floating Point, in una certa quantità di memoria ...  $\geq$  float ... per noi 64 bit ...)
- char** un sottoinsieme dei caratteri alfanumerici; una locazione dedicata a valori di questo tipo e' di solito di 1 byte

# DICHIARAZIONE

PER POTER ESSERE UTILIZZATE, LE VARIABILI **VANNO DICHIARATE** (OPERAZIONE CHE CORRISPONDE A DIRE "UTILizzerò UNA VARIABILE  $x$  DI TIPO **int**" OPPURE "UTILizzerò UNA VARIABILE numeroReale DI TIPO **float**").

**SINTASSI** : **tipo nome**;

**ESEMPI** : **int**  $x$ ; **float** radice; **int**  $y$ , numero, resto;

**SEMANTICA** : LA DICHIARAZIONE SERVE A DIRE AL CALCOLATORE : "UTILizzerò UNA VARIABILE nome DI TIPO **tipo**, QUINDI PREPARA UN CONTENITORE (UN'AREA DI MEMORIA) PER VALORI DI TIPO **tipo** CON NOME nome".

Modello alla Parte Italiana Tipo  
N. Protocollo \_\_\_\_\_  
Data di presentazione \_\_\_\_\_ UNI  
COGNOME \_\_\_\_\_ NOME \_\_\_\_\_  
CODICE FISCALE \_\_\_\_\_

Informazioni sul trattamento dei dati personali ai sensi dell'art. 13 del D.lgs. n. 196 del 30 giugno 2003 in materia di protezione dei dati personali.

Con questo informativo l'Agenzia delle Entrate spiega come utilizza i dati raccolti e quali sono i diritti riconosciuti all'interessato. Infatti, il d.lgs. n. 196/2003, "Codice in materia di protezione dei dati personali", prevede un sistema di garanzia a tutela dei trattamenti che vengono effettuati sui dati personali.

**Finalità del trattamento**  
I dati forniti con questo modello vengono trattati dall'Agenzia delle Entrate esclusivamente per le finalità di liquidazione, accertamento e riscossione delle imposte.  
I dati personali potranno essere comunicati a soggetti pubblici o privati solo nei casi previsti dalle disposizioni del Codice in materia di protezione dei dati personali (art. 19 del d.lgs. n. 196 del 2003). potranno, inoltre, essere pubblicati con le modalità previste dal combinato disposto dell'art. 27 del D.L. n. 600 del 27 settembre 1975, così come modificato dalla legge n. 133 del 6 agosto 2005 e dalla del D.L. n. 83 del 28 gennaio 1972.  
I dati indicati nella presente dichiarazione possono essere trattati anche per l'applicazione dello strumento del c.d. "reddito fisso", compresi i dati relativi alla costituzione del nucleo familiare. I dati trattati ai fini dell'individuazione del nucleo familiare non sono comunicati ai comuni.





# DICHIARAZIONE

LA DICHIARAZIONE DI UNA VARIABILE COMPORTA LA "ALLOCAZIONE" IN MEMORIA CENTRALE DI UNO SPAZIO, DESTINATO A MEMORIZZARE I VALORI ASSUNTI DALLA VARIABILE DURANTE L'ESECUZIONE DEL PROGRAMMA.

LIVELLO LOGICO



LIVELLO FISICO



IL TIPO DI UNA VARIABILE DETERMINA QUALI VALORI LA VARIABILE PUÒ CONTENERE, QUALI OPERAZIONI SI POSSONO EFFETTUARE SULLA VARIABILE E QUANTO SPAZIO VIENE ALLOCATO IN MEMORIA PER LA VARIABILE STESSA.

Le dichiarazioni di variabile ~~andrebbero~~ vanno messe all'inizio del programma, cioè dopo le direttive ("include") e prima di qualunque istruzione.

MI AI PRESO  
IL QUORE  
ANCHE IL LIBRO  
DI GRAMMATIC  
G.N.

# DICHIARAZIONE

LA DICHIARAZIONE DI  
CENTRALE DI UNO SPAZIO  
VARIABILE DURANTE L

LIVELLO LO



IL TIPO DI UNA VARIAB  
QUALI OPERAZIONI SI F  
ALLOCRTO IN MEMORIA



MEMORIA  
SUNTI DALLA

CONTENERE,  
TO SPAZIO VIENE

## Le dichiarazioni di variabile vanno messe all'inizio del programma



(piu` precisamente, all'inizio del blocco di codice in cui sono usate).

# Istruzione di Assegnazione

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

## Sintassi

```
nomeVariabile = espressione;
```

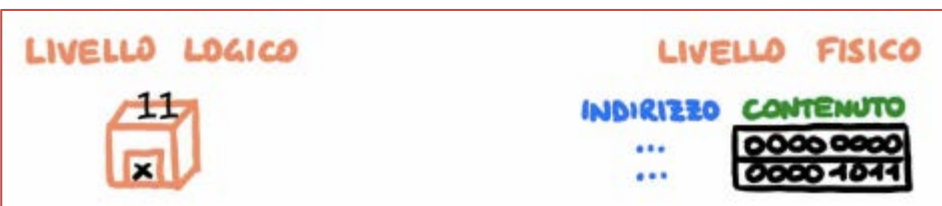
## Esempi

```
x=11;
```

```
y = (x/5) + 27;
```

## Semantica

l'espressione viene valutata ed il valore risultante viene memorizzato nella locazione associata alla variabile



# Istruzione di Assegnazione

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

## Sintassi

```
nomeVariabile = espressione;
```

una **espressione** è una scrittura che può combinare operatori ed operandi, oppure rappresentare l'accesso ad una singola variabile; la «esecuzione» di una espressione si chiama «**valutazione**» e da' luogo ad un valore (di un certo tipo).

## Esempi

```
x=11;  
radiceReale1 = (-b + sqrt(b*b - 4*a*c)) / (2*a);  
y = (x/5) + 27;
```

Anche queste sono espressioni

## Semantica

l'espressione viene valutata ed il valore risultante viene memorizzato nella locazione associata alla variabile

LIVELLO LOGICO



LIVELLO FISICO





# Assegnazione

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

## Sintassi



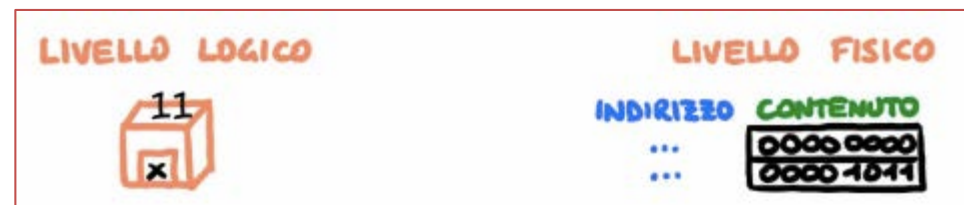
una espressione è una scrittura che può combinare operatori ed operandi, oppure rappresentare l'accesso ad una singola variabile; la «esecuzione» di una espressione si chiama «valutazione» e da' luogo ad un valore (di un certo tipo).

## Esempi

```
x=11; y=(x/5) + 27;  
radiceReale1=(-b + sqrt(b*b - 4*a*c)) / (2*a)
```

## Semantica

l'espressione viene valutata ed il valore risultante viene memorizzato nella locazione associata alla variabile



# Assegnazione

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

Sintassi



Una assegnazione e' una scrittura che può combinare operatori  
ed  
vo  
«  
singola  
na  
po).

Esempi

```
x=11;
```

**Chicca ...**

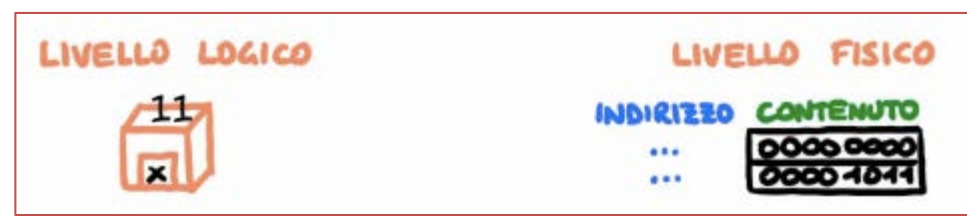
**Operatore di assegnazione**

(  
anche l'istruzione di assegnazione e' una espressione, che viene valutata.  
La sua valutazione corrisponde all'esecuzione dell'assegnazione del  
left\_value al right\_value, e il valore risultante dalla valutazione e' il  
valore che e' stato usato per l'assegnazione  
).

(2\*a)

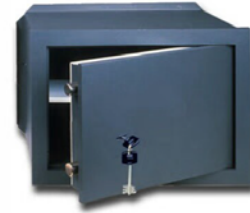
Semantica

l'espressione viene valutata ed il valore risultante viene memorizzato nella locazione associata alla variabile



# ACCESSO

OPERAZIONE CHE PERMETTE DI **ACCEDERE** AL VALORE MEMORIZZATO ALL'INTERNO DI UNA VARIABILE.



**SINTASSI** : nome            OVVERO L'ACCESSO AD UN VALORE MEMORIZZATO DA UNA VARIABILE SI FA SCRIVENDO IL NOME DELLA VARIABILE

**REGOLA** : SE nome COMPARE **A SINISTRA** DI UN OPERATORE DI ASSEGNAZIONE LA VARIABILE VIENE USATA **PER MEMORIZZARE** UN VALORE. ALTRIMENTI, LA VARIABILE VIENE USATA **PER ACCEDERE** AL VALORE CHE MEMORIZZA.

**SEMANTICA** : UTILIZZA IL VALORE MEMORIZZATO IN nome AL POSTO DI nome NELLA PORZIONE DI CODICE IN CUI LA VARIABILE COMPARE.

Esempio:         $y = (x/5) + 27;$

viene acceduto il valore di x; questo valore viene diviso per 5, e si ottiene il valore della sotto-espressione (x/5); a questo valore viene aggiunto 27 e si ottiene il risultato della valutazione **dell'espressione a destra**;

Poi il risultato della valutazione viene memorizzato in y.

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
    int primoNumero, secondo;    /* i due interi */
    int ris;                     /* il risultato */

    primoNumero =168;
    secondo = 640;

    ris = primoNumero + secondo;    /* calcolo */

    printf ("il risultato di %d piu' %d è %d\n",
            primoNumero, secondo, ris);

    return 0;
}
```

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una variabile che viene stampata */
```

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

Programma  
principale

```
int main () {
    int primoNumero, secondo;    /* i due interi */
    int ris;                      /* il risultato */

    primoNumero = 168;
    secondo = 640;

    ris = primoNumero + secondo;    /* calcolo */

    printf ("il risultato di %d piu' %d è %d\n",
            primoNumero, secondo, ris);

    return 0;
}
```

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una variabile, il cui valore poi viene stampata */
```

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

Corpo (body) del  
programma

```
int main () {
```

```
int primoNumero, secondo; /* i due interi */
```

```
int ris; /* il risultato */
```

```
primoNumero = 168;
```

```
secondo = 640;
```

```
ris = primoNumero + secondo; /* calcolo */
```

```
printf ("il risultato di %d piu' %d è %d\n",  
        primoNumero, secondo, ris);
```

```
return 0;
```

```
}
```

{ } racchiudono un  
BLOCCO di istruzioni

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una variabile, il cui valore poi viene stampato */
```

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

DICHIARAZIONI  
DI VARIABILI

```
int main () {
```

```
int primoNumero, secondo;
```

```
/* i due interi */
```

```
/* il risultato */
```

```
primoNumero =168;
```

```
secondo = 640;
```

```
int ris;
```

```
ris = primoNumero + secondo; /* calcolo */
```

```
printf ("il risultato di %d piu` %d e` %d\n",  
        primoNumero, secondo, ris);
```

```
return 0;  
}
```

/\* COMMENTI \*/

Non sono istruzioni, non verranno "eseguite" ... ma sono importanti!!

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una variabile intera, poi viene stampata */
```

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

DICHIARAZIONI

```
int main () {
```

```
int primoNumero
```

```
interi */
```

```
ultato */
```

```
primoNumero = 1
```

```
secondo = 640;
```

```
int ris;
```

```
ris = primoNumero + secondo; /* calcolo */
```

```
printf ("il risultato di %d piu` %d e` %d\n",  
        primoNumero, secondo, ris);
```

```
return 0;
```

```
}
```





# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una variabile intera, poi viene stampata */
```

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

DICHIARAZIONI  
DI VARIABILI

```
int main () {
```

```
int primoNumero, secondo;  
int ris;
```

```
/* i due interi */
```

```
/* il risultato */
```

```
primoNumero =168;
```

```
secondo = 640;
```

```
ris = primoNumero + secondo;          /* calcolo */
```

```
printf ("il risultato di %d piu` %d e` %d\n",  
        primoNumero, secondo, ris);
```

```
return 0;
```

```
}
```

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una terza variabile, che poi viene stampata */
```

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

```
int main () {
```

```
    int primoNumero; /* numero primo */
```

```
    int secondo; /* numero secondo */
```

Istruzioni di assegnazione

```
    primoNumero = 168;
```

```
    secondo = 640;
```

```
    ris = primoNumero + secondo; /* calcolo */
```

```
    printf ("il risultato di %d piu` %d e` %d\n",  
           primoNumero, secondo, ris);
```

```
return 0;
```

```
}
```

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
```

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

```
int main () {
```

```
    int primoNumero, secondo;    /* i due interi */
```

```
    int ris;                      /* il risultato */
```

```
    primoNumero = 168;
```

```
    secondo = 640;
```

Espressione: un operatore e due operandi

```
    ris = primoNumero + secondo;    /* calcolo */
```

```
    printf ("il risultato di %d piu` %d e` %d\n",
```

```
            primoNumero, secondo, ris);
```

```
return 0;
```

```
}
```

Un'espressione combina operatori ed operandi; viene VALUTATA e da' luogo ad un VALORE (il risultato della valutazione)

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
```

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

```
int main () {
```

```
    int primoNumero, secondo;    /* i due interi */
```

```
    int ris;                      /* il risultato */
```

```
    primoNumero = 168;
```

```
    secondo = 640;
```

Pero` anche questa e` un'espressione (640 e` una "costante numerica" inserita nel codice, il cui valore e` ... 640. 640 viene assegnato a secondo.

```
    ris = primoNumero + secondo;    /* calcolo */
```

Un'espressione combina operatori ed operandi;

**oppure** contempla un valore costante;

**oppure, infine**, contempla l'accesso ad una singola variabile ed in questo caso  
(es. ris = primoNumero;)

la valutazione corrisponde all'accesso alla variabile (right value)

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una terza variabile, che poi viene stampata */
```

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

```
int main () {
```

```
    int primoNumero, secondo;    /* i due interi */
```

```
    int ris;                      /* il risultato */
```

```
    primoNumero =168;
```

```
    secondo = 640;
```

```
    ris = primoNumero + s
```

"Istruzione" di stampa

```
*/
```

```
    printf ("il risultato di %d piu` %d e` %d\n",  
           primoNumero, secondo, ris);
```

```
return 0;
```

```
}
```

Veramente si tratta di una **CHIAMATA DI FUNZIONE** (una funzione di libreria - la libreria delle funzioni di I/O standard); e' un **MODULO**, o **sottoprogramma**; il codice di questo sottoprogramma non sta qui, ma nella "libreria oggetto" corrispondente a `stdio.h`. Quando questa funzione viene chiamata, 1) l'esecuzione del programma principale viene sospesa; 2) viene eseguito il codice del sottoprogramma; 3) l'esecuzione del programma principale riprende.

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,  
assegnando il risultato ad una terza variabile, che poi viene stampata */
```

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

```
int main () {  
    int primoNumero;  
    int ris;
```

DIRETTIVA di inclusione, in questo file, del contenuto di `stdio.h`; questo "file header" contiene le dichiarazioni delle funzioni della libreria di I/O standard, tra le quali `printf()`. (non il codice di queste funzioni, quello sta nella libreria oggetto

```
    primoNumero = 1;  
    secondo = 640;
```

```
    ris = primoNumero + secondo;           /* calcolo */
```

```
    printf ("il risultato di %d piu` %d e` %d\n",  
           primoNumero, secondo, ris);
```

```
return 0;  
}
```

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
    int primoNumero, secondo;    /* i due interi */
    int ris;                      /* il risultato */

    primoNumero =168;
    secondo = 640;

    ris = primoNumero + secondo;    /* calcolo */

    printf ("il risultato di %d piu` %d e` %d\n",
            primoNumero, secondo, ris);

    return 0;
}
```

STOP

Anche il programma principale e` una funzione ... la funzione "main".  
Quando la sua esecuzione termina, l'istruzione return passa all'esterno (a  
chi ha chiesto l'esecuzione del programma; al Sistema operativo ...) un  
valore ... di solito zero

# Quale algoritmo rappresenta ourProgram.c ??

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
    int primoNumero, secondo;    /* i due interi */
    int ris;                     /* il risultato */

    primoNumero =168;
    secondo = 640;

    ris = primoNumero + secondo;    /* calcolo */

    printf ("il risultato di %d piu` %d e` %d\n",
            primoNumero, secondo, ris);

return 0;
}
```



# Quale algoritmo rappresenta ourProgram.c ??

- 0) Servono primoNumero e secondo e ris (strutture dati ...)
- 1) Assegna primoNumero con 168
- 2) Assegna 640 a secondo
- 3) Assegna a ris il risultato di primoNumero+secondo
- 4) Stampa ris
- 5) Fine

Come sarebbe il programma corrispondente in linguaggio macchina?

Quale preferiamo scrivere?

Quello in C, esatto.

# Quale algoritmo rappresenta ourProgram.c ??

- 0) Servono primoNumero e secondo e ris (strutture dati ...)
- 1) Assegna primoNumero con 168
- 2) Assegna 640 a secondo
- 3) Assegna a ris il risultato di primoNumero+secondo
- 4) Stampa ris
- 5) Fine

Come sarebbe il programma corrispondente in linguaggio macchina?

Quale preferiamo scrivere?

Quello in C, esatto.

Perché?

Perché scrivere un programma direttamente in LM è difficile, poco naturale per un umano, costringe a gestire direttamente la memoria, ...

# Pregi di un LINGUAGGIO AD ALTO LIVELLO

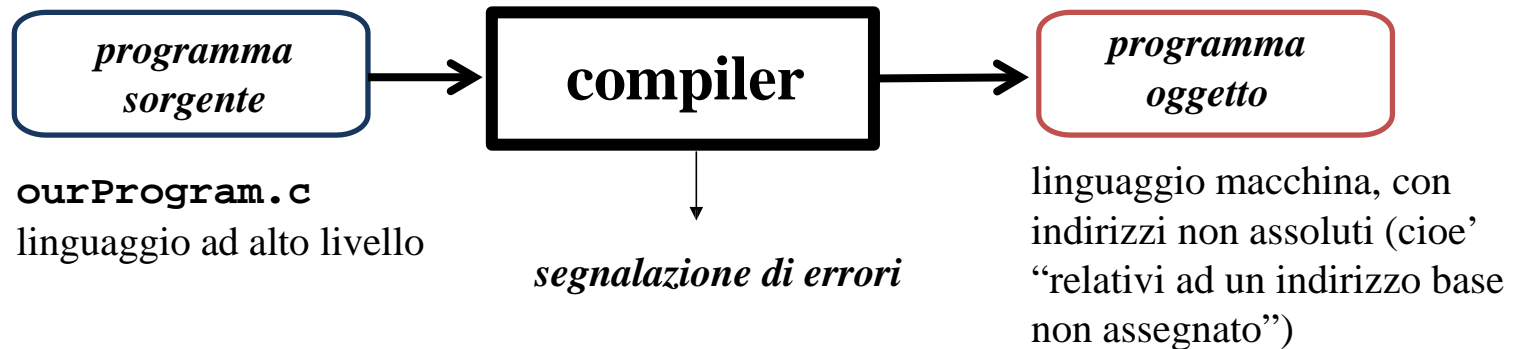
IL C è un linguaggio ad alto livello, termine contrapposto a «basso livello» cioè quello della macchina.

Un programma scritto in LINGUAGGIO AD ALTO LIVELLO,

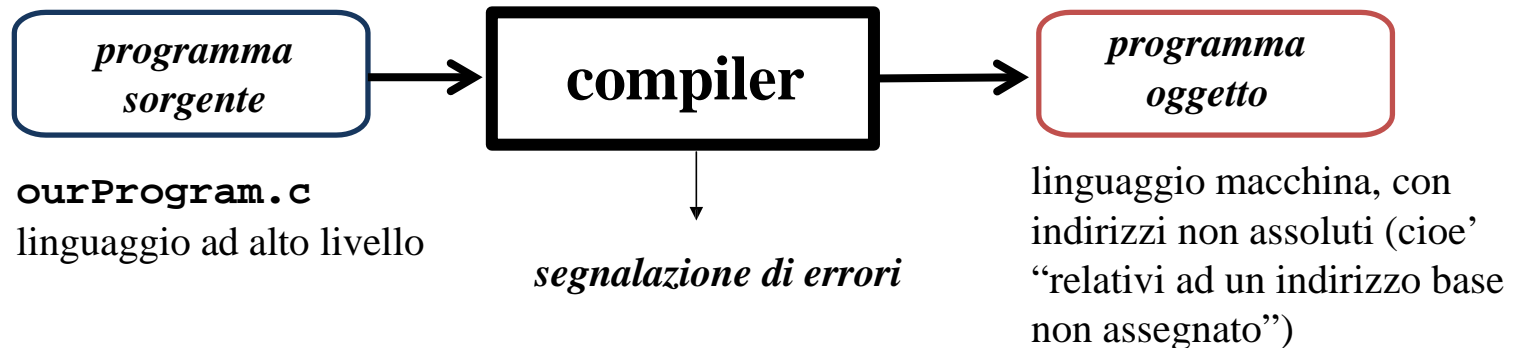
- è più **simile** al linguaggio naturale e al modo in cui pensiamo ai problemi ... ci permette di usare espressioni e istruzioni più simili alle notazioni (matematiche) che usiamo di solito; usiamo parole (le **PAROLE CHIAVE** del linguaggio) che ci sono vicine e comprensibili (**if**, **while**, **for**, **repeat**, **until**, **do** ... non tutte del C) ... o comunque più vicine a noi di quanto non sia **0001000** !!
- **Portabile** (lo stesso programma funziona su qualsiasi macchina, mentre un programma in LM funziona con le CPU che usano quel LM). Come questo succeda si vede tra poco.

**Si', CPU diverse hanno linguaggi diversi ...**

# Dal programma in C al programma eseguito - 1/3



# Dal programma in C al programma eseguito - 1/3



Nel programma ci sono inclusioni di librerie (cioè di insiemi di funzioni, definite da altri ma che ci fanno comodo e che usiamo nel programma («chiamando» quelle funzioni)).

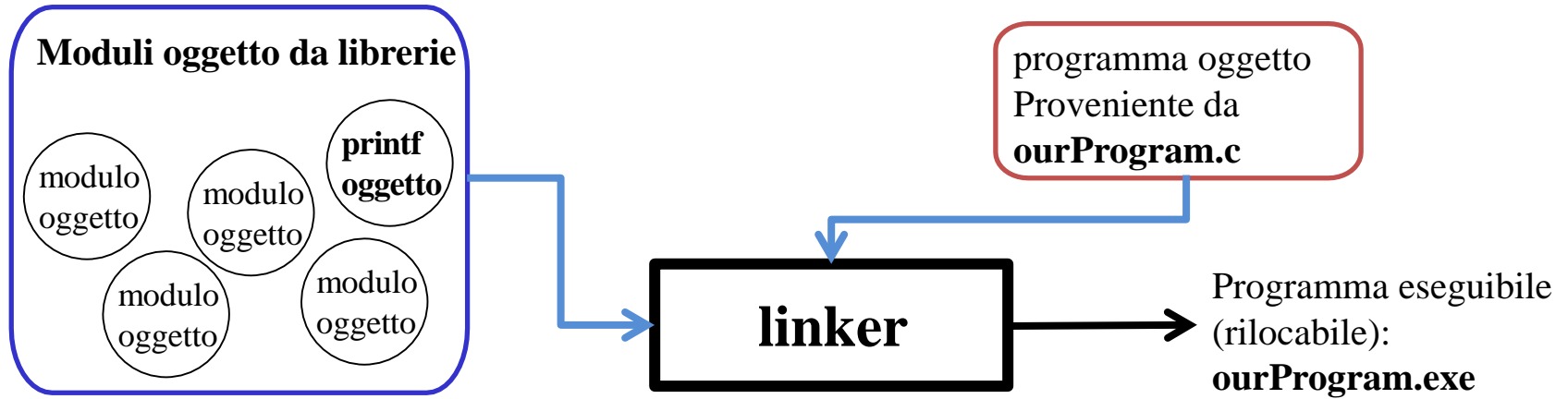
O anche riferimenti a funzioni sviluppate da noi, e che noi vogliamo usare nel programma.

Le funzioni che noi definiamo nel programma (nel file .c) e quelle definite da noi o da altri nelle librerie incluse nel programma, sono anche chiamate «**moduli**» del programma.

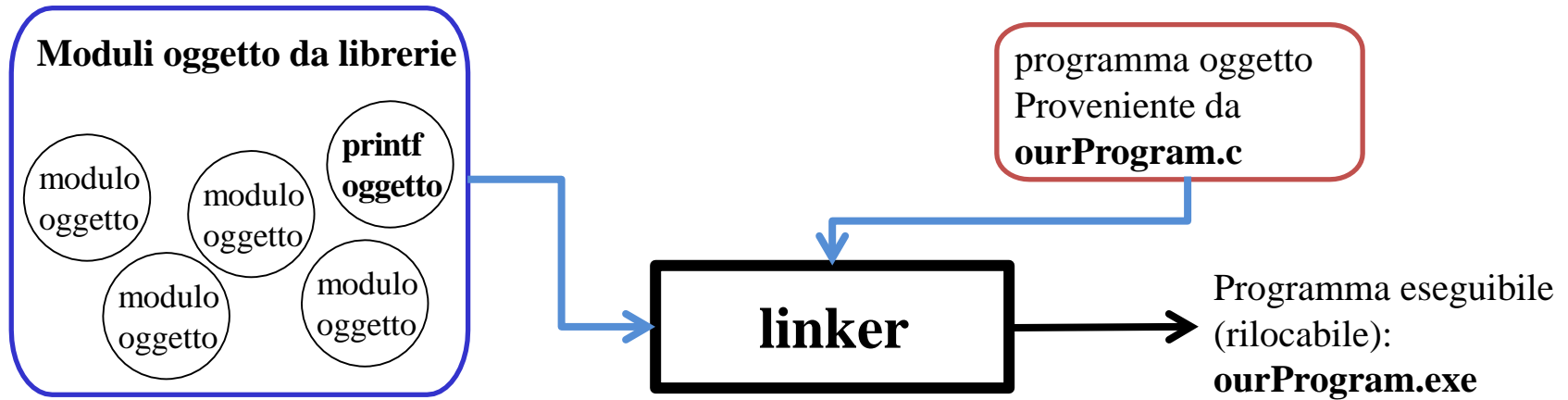
I moduli che abbiamo scritto nel file (più avanti parleremo di questo moduli: le funzioni) hanno codice sorgente che verrà compilato.

I moduli esterni vanno compilati a parte ed uniti poi al risultato della compilazione del nostro programma.

# Dal programma in C al programma eseguito - 2/3



# Dal programma in C al programma eseguito - 2/3



Per ottenere un programma eseguibile bisogna prima unire tutti i moduli oggetto: quello del nostro programma, ottenuto con la sua compilazione (andata bene ...), e quelli delle funzioni esterne, cioè quelle non scritte da noi nel file `ourProgram.c`.

Questa unione si chiama *collegamento* (linking) e viene eseguita quando tutti i moduli sono stati compilati e quindi hanno un codice oggetto disponibile.

Per il nostro programma, la compilazione dobbiamo attivarla noi; invece per le funzioni definite nelle librerie la compilazione è automatica e viene curata dall'ambiente di programmazione in cui lavoriamo (come il DEV).

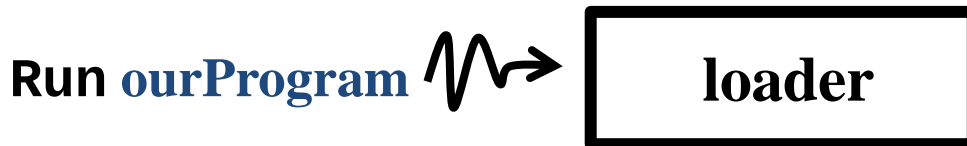
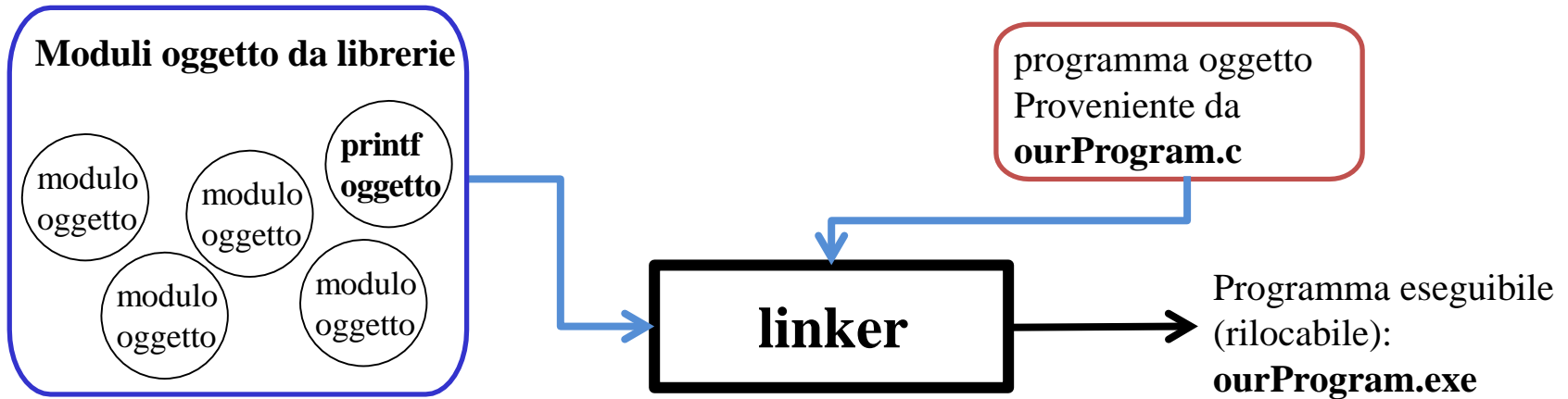
Il risultato del linking è il programma «rilocabile», cioè in sostanza il programma completo di tutte le sue parti, in linguaggio macchina, ma con gli indirizzi che sono tutti definiti rispetto ad un indirizzo base (non ancora noto: sarà noto solo quando effettivamente piazieremo il programma nella memoria – "caricamento", "loading" - quindi per ora si assume zero).

Ora il programma può essere eseguito.

L'indirizzo base viene definito quando il programma viene caricato in memoria centrale (a partire proprio da quel certo indirizzo), cioè al "Run"

**Run ourProgram**

# Dal programma in C al programma eseguito - 3/3



Il programma viene caricato in memoria centrale, a partire da un indirizzo (occupando una zona libera). Ora tutti gli indirizzi usati nel programma oggetto diventano assoluti ed il programma in linguaggio macchina viene eseguito (vedi la parte di architettura, e in particolar modo il processore).



# DICHIARAZIONE CON INIZIALIZZAZIONE

**INIZIALIZZAZIONE** : PRIMA ASSEGNAZIONE DI UN VALORE AD UNA VARIABILE.

**DICHIARAZIONE CON INIZIALIZZAZIONE** : DICHIARA ED INIZIALIZZA UNA VARIABILE NELLA STESSA ISTRUZIONE.

**SINTASSI** : **tipo** nome = espressione ;

**ESEMPIO** : **int** x = 5 ;

**SEMANTICA** : COMBINA LA SEMANTICA DI **DICHIARAZIONE** ED **ASSEGNAZIONE**.

# ourProgram.c

## (usando la inizializzazione in definizione)

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
    int primoNumero=168, secondo=640, ris; /* gli interi ...*/

    ris = primoNumero + secondo;          /* calcolo */

    printf ("il risultato di %d piu` %d e` %d\n",
            primoNumero, secondo, ris);

return 0;
}
```

# STAMPA

LA FUNZIONE **printf** PERMETTE DI STAMPARE UNA **SEQUENZA DI CARATTERI**,  
OVVERO UNA **STRINGA**, CHE DEVE ESSERE SPECIFICATA FRA **DOPPI APICI** "..."

```
printf("STATE IMPARANDO UN SACCO DE ROBBA!");
```



La funzione printf() permette anche di stampare valori, all'interno della stringa di output.

Da adesso la stringa si chiama **stringa di formato**

Per riuscirci, la stringa deve contenere l'indicazione di

- 1) dove la stampa di un valore deve avvenire lungo la stringa di caratteri;
- 2) qual è il **formato di conversione** da usare per stampare quel valore (in pratica di che tipo ci aspettiamo che sia il valore)
- 3) qual è il **valore** ...

La stampa avviene nel punto in cui appare il formato di conversione.

Il formato di conversione inizia con %

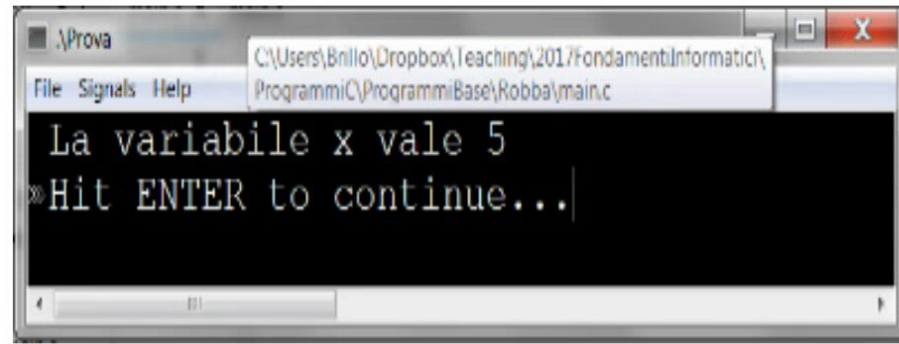
Il valore da stampare appare dopo la stringa di formato, separato da una virgola.

Se ci sono più valori da stampare, devono esserci anche più formati di conversione e ogni valore corrisponde ad uno di essi, in ordine di apparizione.

# ESEMPI DI STAMPE

```
int main() {  
    int x = 5;  
    printf("La variabile x vale %d", x);  
}
```

*d* È IL **FORMATO** DI STAMPA  
PER VALORI DI TIPO **int**

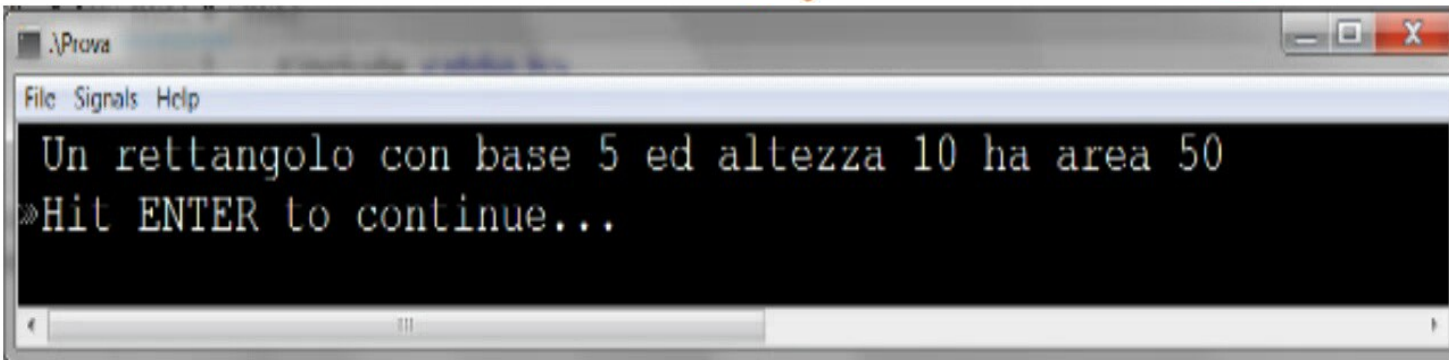


LA **STRINGA** E CIASCUN **VALORE** SONO  
SEPARATI DA VIRGOLE. PER OTTENERE  
IL **VALORE** MEMORIZZATO NELLA VARIABILE  
X SI SCRIVE SEMPLICEMENTE X  
(ACCESSO)

```
int main() {  
    int x = 5;  
    int y = 10;  
    printf("Un rettangolo con base %d ed altezza %d ha area %d", x, y, x*y);  
}
```



I **VALORI** NON  
DEVONO ESSERE  
NECESSARIAMENTE  
VARIABILI, MA  
**ESPRESSIONI** DEL  
TIPO DEL FORMATO



# Caratteri speciali

Alcuni caratteri, espressi con «**sequenze di escape**», realizzate usando backslash '\', hanno significati speciali

<code>\n</code>	andata a capo
<code>\t</code>	tabulazione (un po' di caratteri bianchi ...)
<code>\\</code>	stampa \
<code>\'</code>	stampa `
<code>\"'</code>	stampa doppio apice
<code>\b</code>	back one character

# Lettura, cioè ricezione di dati da INPUT

i dati si "leggono" da INPUT; in Memoria (RAM) i dati memorizzati nelle locazioni si ottengono "accedendo" alle locazioni

LA FUNZIONE **scanf** PERMETTE DI **LEGGERE VALORI** IMMESSI DALL'UTENTE.

**SINTASSI**: **scanf** ("%FORMATO", &VARIABILE);

- **%FORMATO** INDICA IL **FORMATO** DEI VALORI DA LEGGERE (AD ES. **%d** PER LEGGERE UN INTERO) COME PER **printf**
- **&VARIABILE** INDICA L'**INDIRIZZO** DELLA MEMORIA AL QUALE SI TROVA LA VARIABILE

**SEMANTICA**: IL PROGRAMMA SI PONE IN ATTESA CHE L'UTENTE INTRODUCA UN VALORE CON FORMATO **d**. QUANDO L'UTENTE INTRODUCE UN VALORE, TALE VALORE VIENE **MEMORIZZATO NELLA VARIABILE** IL CUI INDIRIZZO È **&VARIABILE**.

# sempre sulla lettura ...

ANALOGAMENTE AD UNA ISTRUZIONE DI ASSEGNAZIONE, UNA LETTURA HA COME EFFETTO QUELLO DI MEMORIZZARE UN VALORE IN UNA VARIABILE.

PER COMUNICARE ALL'UTENTE CHE DEVE IMMETTERE DEI VALORI, UNA ISTRUZIONE DI LETTURA È IN GENERE PRECEDUTA DA UNA ISTRUZIONE DI STAMPA.

```
/* questo programma, ricevendo in INPUT i lati significativi di un rettangolo,
   ne calcola e stampa in OUTPUT l'area */
#include <stdio.h>

int main () {
    int primoLato, secondoLato,          /* i lati da ricevere via INPUT */
        area;                          /* risultato ... */

    /* lettura dei dati */
    printf (" Oh utente, forniscimi gentilmente ..., che cosi' ci lavoro:\n ");
    scanf("%d %d", &primoLato, &secondoLato);

    /* calcolo dell'area */
    area = primoLato*secondoLato;      /* calcolo */

    printf (" ... di lati %d e %d ha area %d\n", primoLato, secondoLato, area);

    printf ("\nFINE");
    return 0;
}
```

## ourProgram algorithm ... without input ...

- 0) Servono primoNumero e secondo e ris (struttura dati ...)
- 1) Assegna primoNumero con 168
- 2) Assegna 640 a secondo
- 3) Assegna a ris il risultato di primoNumero+secondo
- 4) Stampa ris
- 5) Fine

## ... and with input

Versione con INPUT ???



## ourProgram algorithm ... without input ...

- 0) Servono primoNumero e secondo e ris (struttura dati ...)
- 1) Assegna primoNumero con 168
- 2) Assegna 164 a secondo
- 3) Assegna a ris il risultato di primoNumero+secondo
- 4) Stampa ris
- 5) Fine

## ... and with input

Versione con INPUT ???

- 0) (primoNum, secondoNum, ris ... i dati che usiamo)
- 1) INPUT primoNum, secondoNum
- 2) ris = primoNum + secondoNum
- 3) OUTPUT ris
- 4) Fine

→ Vogliamo fare il programma?

# ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da input. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */
```

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

```
int main () {
```

```
    int primoNumero, secondoNumero,      /* i dati da input */
        ris;                               /* risultato della somma */
```

passo 0)

```
return 0;
```

```
}
```

# ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da input. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */
```

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

```
int main () {
```

```
    int primoNumero, secondoNumero,    /* i dati da input */
        ris;                            /* risultato della somma */
```

```
    printf ("Oh utente, scrivi due numeri e io li sommo: ");
    scanf ("%d %d", &primoNumero, &secondoNumero);
```

```
return 0;
```

```
}
```

passo 1) (chiedere per avere e' intelligente ...)

# ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da input. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */
```

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

passo 2)

```
int main () {
```

```
    int primoNumero, secondoNumero,    /* i dati da input */
        ris;                            /* risultato della somma */
```

```
    printf ("Oh utente, scrivi due numeri e io li sommo: ");
    scanf ("%d %d", &primoNumero, &secondoNumero);
```

```
    ris = primoNumero + secondoNumero;    /* calcolo */
```

```
return 0;
```

```
}
```

# ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da input. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */
```

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

passo 3)

```
int main () {
```

```
    int primoNumero, secondoNumero,      /* i dati da input */
        ris;                               /* risultato della somma */
```

```
    printf ("Oh utente, scrivi due numeri e io li sommo: ");
    scanf ("%d %d", &primoNumero, &secondoNumero);
```

```
    ris = primoNumero + secondoNumero;    /* calcolo */
```

```
    printf (" Stimato user, %d piu` %d e` uguale a %d\n",
            primoNumero, secondoNumero, ris);
```

```
return 0; 
```

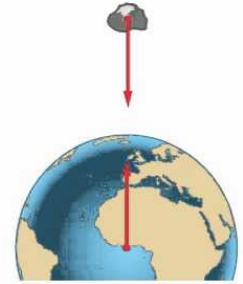
passo 4)

```
}
```

# Costanti



$$F = G \frac{m_1 m_2}{r^2}$$
$$G = 6,67 \cdot 10^{-11} \text{ N} \cdot \frac{\text{m}^2}{\text{kg}^2}$$



Una costante e` un **IDENTIFICATORE** che viene associate ad un **VALORE**

prima della compilazione del programma, e che successivamente manterra` quel valore per tutta la durata dell'esecuzione del programma.

conveniente se quel valore appare tante volte nel programma ...

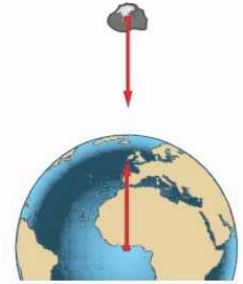
e poi magari serve che sia diverso in una versione lievemente diversa del programma.

```
#define PI 3.14159
```

# Costanti



$$F = G \frac{m_1 m_2}{r^2}$$
$$G = 6,67 \cdot 10^{-11} \text{ N} \cdot \frac{\text{m}^2}{\text{kg}^2}$$



Una costante e` un **IDENTIFICATORE** che viene associate ad un **VALORE**

prima della compilazione del programma, e che successivamente manterra` quel valore per tutta la durata dell'esecuzione del programma.

Ogni volta che, nel programma, viene usato l'identificatore di quella costante, e` il valore corrispondente che viene usato.

L'uso di una costante e` conveniente quando un certo valore deve essere scritto esplicitamente in un programma numerose volte, e magari da questo programma si potrebbe ottenere un altro programma solo variando il valore della costante.

```
#define PI 3.14159
```

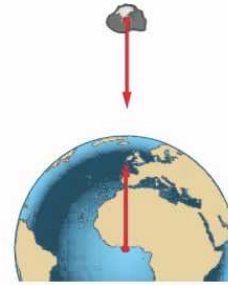
**Esercizio ...**

```
/* programma che chiede e legge da input i  
raggi di tre circonferenze e produce per  
ciascuna circonferenza la relativa area */
```

Vedi il programma `cerchi.c` ... nella nostra geometria PI ha un certo valore ( $\pi$ ) e lo usiamo per calcolare l'area di tre circonferenze, usando PI ripetutamente. Il medesimo programma potrebbe essere usato in una geometria diversa, in cui PI e' diverso, semplicemente cambiando in una sola riga del programma il suo valore... invece che cambiarlo in tutte le istruzioni in cui e' usato.

# Costanti e MACRO

$$F = G \frac{m_1 m_2}{r^2}$$
$$G = 6,67 \cdot 10^{-11} \text{ N} \cdot \frac{\text{m}^2}{\text{kg}^2}$$



$$y = f(x) = c$$

$$y' = \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} =$$

$$= \lim_{h \rightarrow 0} \frac{c - c}{h} = 0$$

$$y = c \rightarrow y' = 0$$

Una costante e' un **IDENTIFICATORE**

che viene associate ad un **VALORE**

prima della compilazione del programma, e che successivamente manterra' quel valore per tutta la durata dell'esecuzione del programma.

## #define PI 3.14159

- Le costanti vanno definite all'inizio del file con il programma, tipicamente prima della main()
- Gli identificatori sono per convenzione scritti con **TUTTE MAIUSCOLE**
- **#define** e' un esempio di direttiva per il compilatore (o **MACROISTRUZIONE**, o direttamente **MACRO**)

Le direttive, come **#define** e **#include**, vengono gestite dal **"precompilatore"**; la pre-compilazione e' una fase precedente alla compilazione, in cui in sostanza si prepara il file.c per essere compilato.



per casa ...

# Quale algoritmo rappresenta ourProgram.c ??

il programma ourProgram.c è nella slide successiva  
... non c'è bisogno di risalire indietro ...

scrivere l'algoritmo completo che corrisponde al programma.

Poi, sì, si può risalire alla slide che abbiamo discusso a lezione e  
confrontare quel che si è scritto

# ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
    int primoNumero, secondo;    /* i due interi */
    int ris;                     /* il risultato */

    primoNumero =168;
    secondo = 640;

    ris = primoNumero + secondo;    /* calcolo */

    printf ("il risultato di %d piu' %d e' %d\n",
            primoNumero, secondo, ris);

    return 0;
}
```

# ourProgram.c

## (usando la inizializzazione in definizione)

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
    int

completare ...

facendo uso dell'inizializzazione
in definizione

l'algoritmo e' sempre il medesimo!
Cambia solo la tecnica con cui plo
programmiamo ...

return 0;
}
```

# esercizi

Nella directory dei complementi didattici per la lezione 03, ci sono risorse per usare il DevC++.

C'e' un "Primer" per iniziare ad usare il DEVC++. Seguitelo. Quando lo avrete terminato sarete un po' piu' pratici della cosa. E potrete scrivere altri programmi.

Ricordatevi di salvare sempre i vostri file (se sono programmi in C) con estensione .c (non .cpp).

Dopo il primer potete dare un'occhiata all'ulteriore file pdf disponibile nella directory dei complementi didattici per la lezione 03

Poi lasciate questa directory, per ora e fate gli esercizi suggeriti nelle prossime slide (le cui "soluzioni" sono sempre nella directory dei complementi didattici per questa lezione)

# esercizi sui rettangoli e sui cerchi

scrivere `rettangolo.c`, senza consultare il file `rettangolo.c` nella directory dei complementi didattici, e nemmeno la slide in cui lo abbiamo discusso.

La specifica del problema e' la seguente:

```
/* questo programma, ricevendo in INPUT i lati significativi
   di un rettangolo, ne calcola e stampa in OUTPUT l'area
   */
```

Provare il programma eseguendolo per almeno 5 rettangoli.

In `rettangolo.c` usare un'unica operazione di INPUT, per leggere i lati.

scrivere `rettangolo2.c`, identico al precedente, a parte che esegue due distinte operazioni di input, una per il primo lato e una per il secondo.

scrivere un

```
/* programma che chiede e legge da input i raggi di tre circonferenze
   e produce per ciascuna circonferenza la relativa area */
```

Usare una costante `PI` per il  $\pi$

POI confrontare la propria soluzione con quella nella directory dei complementi didattici. E correggere quest'ultima che stampa un po' male le cose ...