# Fondamenti di Informatica T-1
# Modulo 2



## Obiettivi di questa esercitazione

1. Esericizi sempici su funzioni
2. Funzioni ricorsive
3. Funzioni e Header File

# ESERCIZIO 1
## (Funzioni)

Codificare in C la funzione

```
int max(int x, int y)
```

che restituisce il massimo valore tra due interi.

Codificare in C la funzione

```
int max3(int x, int y, int z)
```

che restituisce il massimo valore fra tre interi, sfruttando la funzione max definita precedentemente.

Definire un possibile main che prende in ingresso i tre valori dall'utente e ne stampa il massimo.

# ESERCIZIO 1 - Soluzione
## (Funzioni)

```
int max(int a, int b)
{ if (a>b) return a;
    else return b;
}

int max3(int a, int b, int c)
{ int max_di_due;
    max_di_due = max(a,b);
    return max(max_di_due,c);
}
```

# ESERCIZIO 1 – Soluzione - Variante
## (Funzioni)

```
int max3(int a, int b, int c)
{
      if (max(a,b) > c)
            return max(a,b);
      else
            return c;
}
```

```
int max3(int a, int b, int c)
{
      return max(max(a,b),c);
}
```

# ESERCIZIO 1 - Soluzione
## (Funzioni)

Un possibile main

```
int main()
{int MAX;
 int v1, v2, v3;
 printf("Inserisci tre interi");
 scanf("%d, %d, %d", &v1,&v2,&v3);
 MAX = max3(v1,v2,v3);
 printf("Massimo valore inserito: %d",MAX);
 return 0;
}
```

NOTA: Prima di chiamare una funzione è necessario definirla. Nel file sorgente quindi prima del **main** e' necessario definire la funzione **max3** e prima di **max3** bisogna definire **max**

# ESERCIZIO 2
## (Funzioni)

Si scriva una funzione

```
int somma2(int n);
```

che dato `n` deve calcolare $\sum_{i=1}^{n} \sum_{j=1}^{i} j$

A tal fine si sfrutti una funzione

```
int somma(int n);
```

che dato `n` deve calcolare $\sum_{k=1}^{n} k$

# ESERCIZIO 3
## (Funzioni)

Si scriva una funzione

```
int somma_potenze(int a,int n);
```

che dati `a` e `n` deve calcolare $\sum_{i=1}^{n} a^i$

A tal fine si scriva una funzione

```
int potenza(int x,int y);
```

che dati `x` e `y` deve calcolare $x^y$ usando come operazione primitiva il prodotto.

# ESERCIZIO 4
## (Funzioni)

Creare una funzione **`float square(float x)`** . La funzione deve restituire il quadrato del parametro **`x`**.

Creare un'altra funzione, di nome **`float cube(float x)`**, che restituisce invece il cubo del valore **`x`**.

Progettare quindi e codificare un programma che legge un float da tastiera e restituisce il suo quadrato ed il suo cubo. Per calcolare il quadrato ed il cubo si devono utilizzare le due funzioni sopra definite.

# ESERCIZIO 5
## (Funzioni)

Si progettino e si realizzino due funzioni così definite:

```
float euro_to_dollari(float money)
float euro_to_lire(float money)
```

ognuna delle quali converte un valore in euro nella moneta corrispondente. A tal fine si supponga che:
1 € = 0.98 $
1 € = 1936.27 £

Si progetti poi un programma che legge da input un valore intero, inteso come quantità di euro, e stampa la conversione in dollari ed in lire.

# ESERCIZIO 6
## (Funzioni)

Codificare in C la funzione
`int min_to_sec(int a)` che considera il parametro a come minuti e restituisce il numero di secondi corrispondente.

Codificare in C la funzione
`int ore_to_sec(int a)` che considera il parametro a come ammontare di ore, e restituisca il numero di secondi corrispondente. Si utilizzi la funzione definita precedentemente.

Definire un possibile main che prende in ingresso tre valori interi, rappresentanti ore, minuti e secondi della durata di un CD Audio. Il programma deve stampare il valore corrispondente in secondi.

# ESERCIZIO 7
## (Funzioni)

Codificare in C la funzione
`int ipotenusa(int a, int b)` che, dati i cateti `a` e `b` di un triangolo rettangolo, restituisce il valore dell'ipotenusa.

A tal scopo si utilizzi il Teorema di Pitagora:

$$Ipotenusa = \sqrt{a^2 + b^2}$$

Per calcolare la radice quadrata si utilizzi la funzione di libreria `sqrt(x)`. Per utilizzare quest'ultima si aggiunga l'istruzione `#include <math.h>` in testa al file.

Definire un possibile main che legga da tastiera due valori che rappresentino i cateti di un triangolo rettangolo, e stampi il valore dell'ipotenusa.

# ESERCIZIO 8
## (Funzioni)

Codificare in C la funzione
`int perimetro(int a, int b, int c)` che, dati i lati a,b,c di un triangolo, ne calcola il perimetro.

Codificare in C la funzione
`float area(int a, int b, int c)`
che restituisce l'area di un triangolo i cui lati misurano a, b, c. A tal scopo si usi la formula di Erone:

$$Area = \sqrt{p(p-a)(p-b)(p-c)}$$

Dove p è la metà del perimetro. A tal scopo si includa l'header `<math.h>` e si utilizzi la funzione `sqrt(x)`.

Definire un possibile main che prende in ingresso i tre lati di un triangolo e stampa perimetro ed area.

# ESERCIZIO 9
## (Funzioni)

Codificare in C la funzione `int primo(int x)` che restituisce:
1 se x è un numero primo
0 altrimenti.
Si utilizzi a tal proposito l'operatore modulo (%).

Si progetti un programma che legge da tastiera un numero N, e stampa a video tutti i numeri primi compresi tra 0 e N.

# Esercizio 1
(Funzioni ricorsive)

Scrivere una funzione ricorsiva:

int ric(int x)

che calcoli, ricorsivamente, la somma di tutti i numeri compresi tra 0 ed x.

# Esercizio 1 - Soluzione
(Funzioni ricorsive)

```
int ric(int x) {
  if (x == 0)
     return 0;
  else
     return x + ric(x-1);
}
```

# Esercizio 2

(Funzioni ricorsive)

Si scrivano le versioni ricorsiva ed iterativa (utilizzo di while) di una funzione:

**double f(double a, int n);**

che calcoli il seguente valore:

$$\sum_{i=1}^{n}\left(a-\frac{i}{a}\right)$$

# Esercizio 3

(Funzioni ricorsive)

Si scriva una programma che inverta le cifre di un numero intero N usando una funzione apposita. A tal fine, si realizzi sia una versione ricorsiva, sia una versione iterativa della funzione.

Per esempio:

dato N=4325, il programma stampa: 5234

# Esercizio 4
(Funzioni ricorsive)

Si scriva una programma che legga da input una sequenza di caratteri terminati dal tasto “invio”, e stampi a video tale sequenza in ordine invertito. Il programma stampi a video anche il numero di caratteri inseriti.

A tal fine, si realizzi tale funzionalità tramite una funzione ricorsiva.

Per esempio:

se inserito “abcdef<INVIO>”, il programma deve stampare: “fedcba 6”

# Programmi su più moduli - Esempio

## Calcolo del logaritmo in base qualunque

- Incapsulare la logica di calcolo in una funzione
  - PASSO 1: definisco la dichiarazione della funzione (nome, parametri di input e di output)

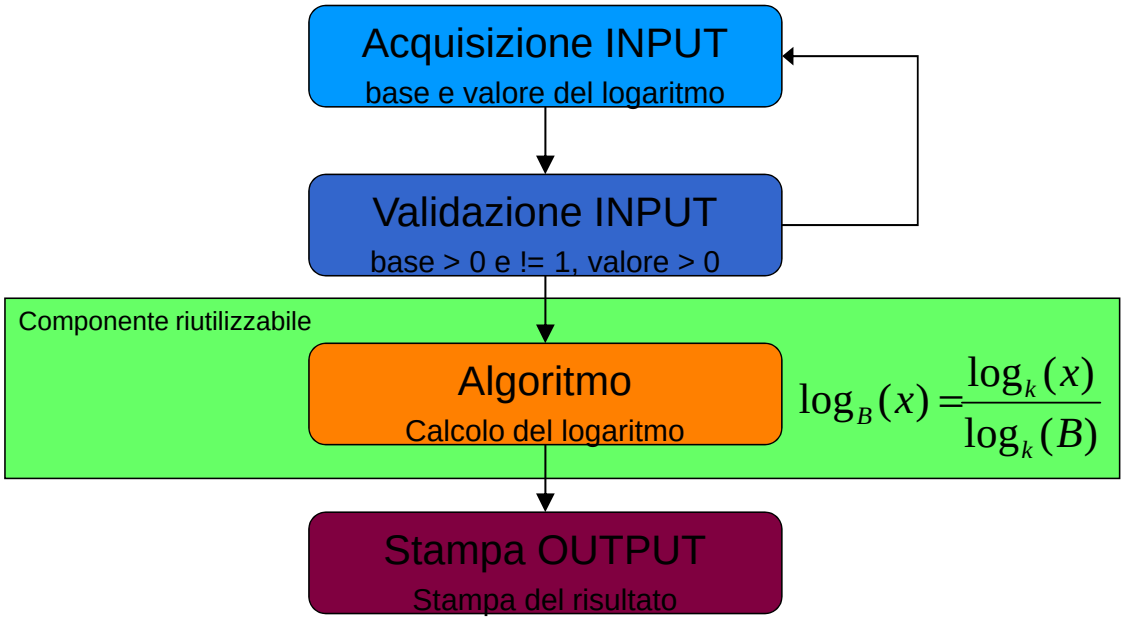
```
float mylog(float base, float value)
```

  - PASSO 2: realizzo la funzione

$$\log_B(x) = \frac{\log_k(x)}{\log_k(B)}$$

# Programmi su più moduli - Esempio

Calcolo del logaritmo in base qualunque - schema di soluzione

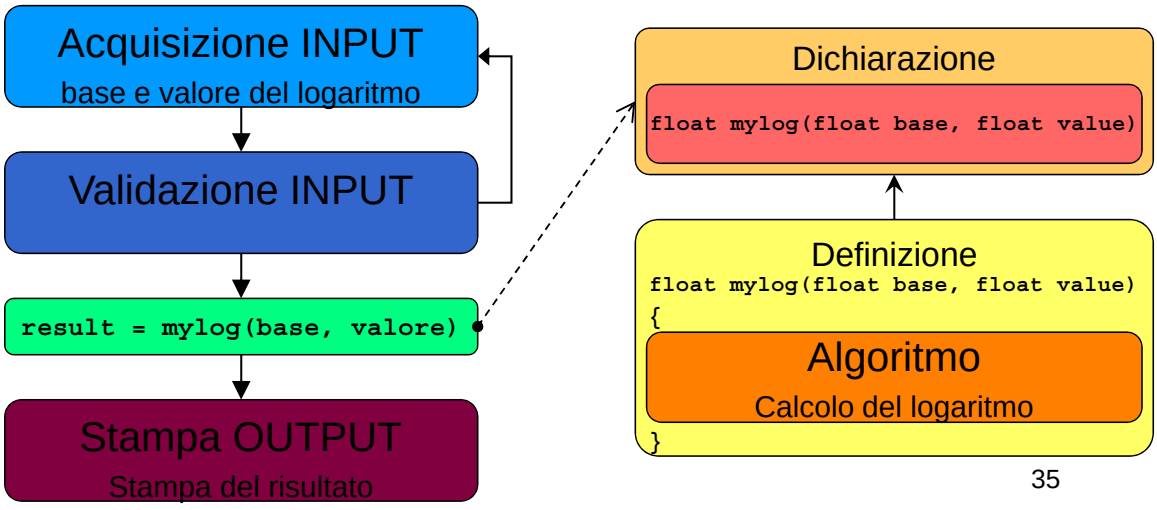





# Programmi su più moduli - Esempio

- Logaritmo come componente: uso di una *funzione*!
  - PASSO 1: **dichiarazione della funzione (nome, parametri di input, parametri di output)**

    ```
    float mylog(float base, float value)
    ```
  - PASSO 2: **definizione della funzione** (ovvero implementazione)

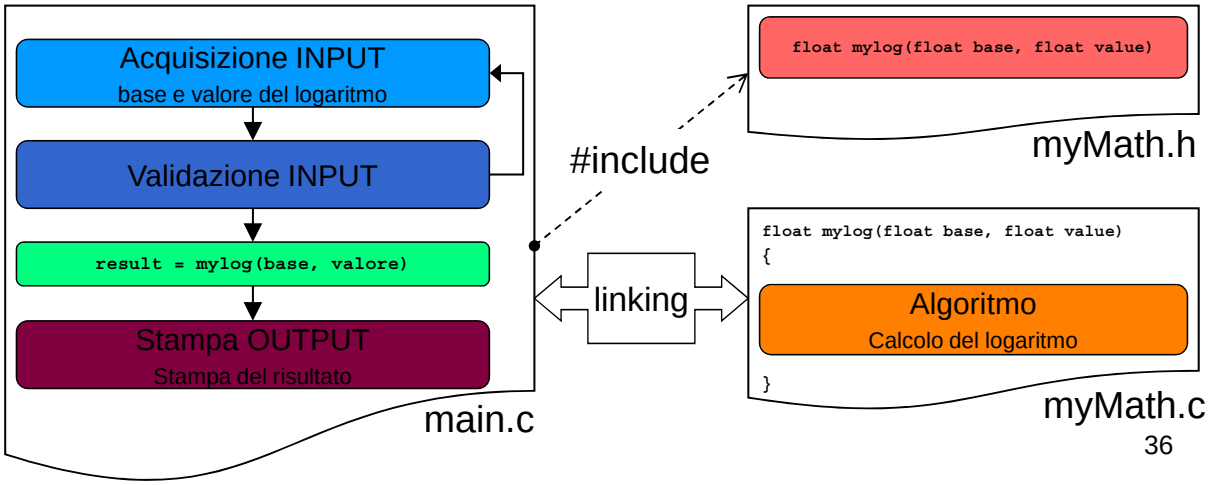


# Programmi su più moduli - Esempio

Vogliamo rendere la funzione **mylog** davvero utilizzabile da più utenti in più programmi: creazione di un ***modulo apposito***

- header file contenente le dichiarazioni (ad es. “myMath.h”)
- file C contenente le definizioni (ad es. “myMath.c”)
- includiamo “myMath.h” nel modulo che contiene la funzione main
- compiliamo con l’istruzione opportuna:

```
cl myProg.c myMath.c /I myMath.h /o myProg.exe
```






# Programmi su più moduli - Esempio

myMath.h:

```
float mylog(float base, float value);
```

myMath.c:

```
#include <math.h>

float mylog(float base, float value)
{
  return log(value) / log(base);
}
```

In realtà, dovrebbe contenere anche la validazione dei dati in input (*mai fidarsi del cliente!*) e restituire **errore** in caso di input non corretto

# Programmi su più moduli - Esempio

main.c:

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


int main ()
{
  double b, x, result;
  …
  result = mylog(b, x);
  …
  return 0;
}
```



# Esercizio 1

(Funzioni e programmi su più moduli)

## Ciclo per il calcolo del massimo e del minimo

- Realizzare un programma che calcoli il minimo e il massimo di una serie di valori
- Il numero di valori deve essere costante e definito tramite una opportuna *costante simbolica*
- Se la differenza tra il massimo e il minimo supera 10, il programma termina, altrimenti aspetta una nuova serie di valori
- Incapsulare ***il calcolo del minimo e del massimo in funzioni apposite, definite in un apposito modulo***

# Esercizio 1
## (Funzioni e programmi su più moduli)

- Quanti cicli? 2
  - Uno esterno per capire se uscire dal programma o richiedere la serie di valori
  - Uno interno per acquisire i K valori
- Che tipo di cicli?
  - Ciclo esterno: verifica una condizione a posteriori
    ➔ do…while
  - Ciclo interno: conosce a priori il numero di iterazioni
    ➔ for
- Di quanti valori devo tener traccia?
  - Ricordarsi che il minimo ed il massimo si possono calcolare passo passo
- Quando devo re-inizializzare il massimo ed il minimo?



# Esercizio 2
## (Funzioni e programmi su più moduli)

**Calcolo del mcm tra numeri interi**

- Realizzare un programma che prenda in input una serie di numeri interi, calcolando via via il minimo comune multiplo tra essi; il programma deve terminare quando mcm diventa più grande di 100
  - Ricordarsi che, come per il massimo e il minimo, anche mcm si può calcolare in modo parziale
  - Quindi basta utilizzare, per il calcolo, il valore di mcm calcolato al passo precedente e il numero inserito al passo corrente

    mcm(a, b, c) = mcm(mcm(a, b), c)

# Esercizio 2
## (Funzioni e programmi su più moduli)

- Utilizzare la relazione $mcm(a,b) = \frac{a \cdot b}{MCD(a,b)}$
- Utilizzare l'algoritmo di Euclide per il MCD tra due numeri
  - Finché M≠N:
    - se M>N, sostituisci a M il valore M' = M-N
    - altrimenti sostituisci a N il valore N' = N-M
    - MCD è il valore finale ottenuto quando M e N diventano uguali
- Incapsulare il calcolo di mcm e MCD in due funzioni
  - Ragionare per astrazione!
    - Individuare prima come le funzioni vengono viste dall'esterno (dichiarazione), poi realizzarle



# Esercizio 2
## (Funzioni e programmi su più moduli)

- Procedere per passi
  - Prima definiamo la funzione per MCD e proviamo a testarla su due valori
  - Poi definiamo la funzione per mcm e proviamo a testarla su due valori
  - Poi realizziamo il programma ciclico
- Per ultimo, utilizziamo l'approccio a moduli inserendo il calcolo di MCD e mcm in un modulo di libreria
  - *Header File contenente le dichiarazione delle funzioni*

# Esercizio 2
(Funzioni e programmi su più moduli)

- Esempio di esecuzione

```
Inserisci il primo valore: 4
Inserisci un valore: 8
mcm corrente: 8
Inserisci un valore: 12
mcm corrente: 24
Inserisci un valore: 10
mcm corrente: 120
```



# Esercizio 3
(Funzioni e programmi su più moduli)

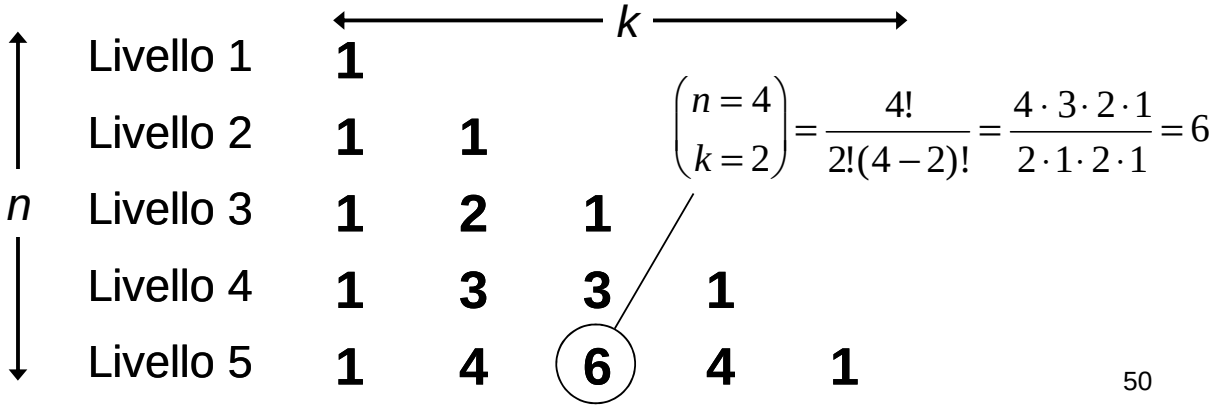
## Triangolo di Tartaglia

- Realizzare un programma che, letto in input il massimo livello voluto, mostri a video il contenuto del triangolo di Tartaglia fino a quel livello
- Per la costruzione del triangolo di Tartaglia, si utilizzi la corrispondenza tra i suoi elementi e i coefficienti binomiali

# Esercizio 3
## (Funzioni e programmi su più moduli)

- Coefficiente binomiale $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

- Triangolo di Tartaglia (5 livelli)
  - Allineato a sinistra per semplicità di stampa






# Esercizio 3
## (Funzioni e programmi su più moduli)

Organizzare il programma in due moduli separati

- I modulo (di libreria)
  - Funzione che calcola il fattoriale
    - Fattoriale di 0 = 1
    - Fattoriale di N = prodotto dei numeri da 1 a N
  - Funzione che calcola il coefficiente binomiale
    - A partire dalla funzione che calcola il fattoriale
  - Prima header file
- II modulo (main)
  - Acquisizione in input del numero dei livelli
  - Stampa del triangolo di Tartaglia
    - Come utilizzare i cicli? Quanti cicli sono? Che tipo di cicli?
    - Ricordarsi che il coefficiente binomiale è definito solo per k≤n