

Fondamenti di Informatica T-1

Modulo 2

Obiettivi di questa esercitazione

1. Esercizi semplici su funzioni
2. Funzioni ricorsive
3. Funzioni e Header File

ESERCIZIO 1

(Funzioni)

Codificare in C la funzione

```
int mioMax(int x, int y)
```

che restituisca il massimo valore tra due interi.

Codificare in C la funzione

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

che restituisca il massimo valore fra tre interi, sfruttando la funzione `miomax` definita precedentemente.

Definire un possibile `main` che prenda in ingresso i tre valori dall'utente e ne stampi il massimo.

ESERCIZIO 1 - Soluzione

(Funzioni)

```
int mioMax(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 = mioMax(a,b);
  return mioMax(max_di_due,c);
}
```

ESERCIZIO 1 – Soluzione - Variante

(Funzioni)

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

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

ESERCIZIO 1 - Soluzione

(Funzioni)

Un possibile main

```
int main()
{int theMax;
  int v1, v2, v3;
  printf("Inserisci tre interi");
  scanf("%d, %d, %d", &v1, &v2, &v3);
  theMax = max3(v1, v2, v3);
  printf("Massimo: %d", theMax);
  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

`theMax`

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 2 - Soluzione

(Funzioni)

```
int somma(int n)
{ int k, s=0;
  for (k=1; k<=n; k++)
    s = s + k;
  return s;
}

int somma2(int n)
{int i, s2 = 0;
  for (i=1; i<=n; i++)
    s2 = s2 + somma(i);
  return s2;
}
```

NOTA: Nel file sorgente prima del `main` e' necessario definire la funzione `somma2` e prima di `somma2` bisogna definire `somma`

ESERCIZIO 2 - Soluzione

(Funzioni)

Un possibile main

```
int main() {
    int N, S;

    printf("Inserisci un intero");
    scanf("%d", &N);

    S = somma2(N);
    printf("La somma vale %d", S);

    return 0;
}
```

ESERCIZIO 3

(Funzioni)

Si scriva una funzione

```
int sommaPotenze(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 3 - Soluzione

(Funzioni)

```
int potenza(int x,int y)
{ int i, P=1; /* P: accumulatore di prod.*/
  for(i=1; i<=y; i++)
    P = P * x;
  return P;
}
```

```
int sommaPotenze(int a, int n)
{ int i, s=0;
  for(i=1; i<=n; i++)
    s = s + potenza(a,i);
  return s;
}
```

ESERCIZIO 3 - Soluzione

(Funzioni)

Un possibile main

```
int main() {
    int N1,N2,SP;

    printf("Inserisci due interi");
    scanf("%d,%d", &N1,&N2);

    SP = sommaPotenze(N1,N2);
    printf("La somma delle potenze vale %d",SP);

    return 0;
}
```

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 restituisca invece il cubo del valore `x`.

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

ESERCIZIO 4 - Soluzione

(Funzioni)

```
float square(float x)
{
    return x*x;
}
```

```
float cube(float x)
{
    return x*x*x;
}
```

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 legga da input un valore intero, inteso come quantità di euro, e stampi la conversione in dollari ed in lire.

ESERCIZIO 5 - Soluzione

(Funzioni)

```
float euro_to_dollari(float money)
{
    return money*0.98;
}
```

```
float euro_to_lire(float money)
{
    return money*1936.27;
}
```


ESERCIZIO 6

(Funzioni)

Codificare in C la funzione

`int min_to_sec(int a)` che consideri il parametro `a` come minuti e restituisca il numero di secondi corrispondente.

Codificare in C la funzione

`int ore_to_sec(int a)` che consideri 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 prenda 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 6 - Soluzione

(Funzioni)

```
int min_to_sec(int a)
{
    return a*60;
}
```

```
int ore_to_sec(int a)
{
    return a*60*60;
}
```

ESERCIZIO 7

(Funzioni)

Codificare in C la funzione

`float ipotenusu(float a, float b)` che, dati i cateti `a` e `b` di un triangolo rettangolo, restituisca il valore dell'ipotenusa.

A tal scopo si utilizzi il Teorema di Pitagora:

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

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 7 - Soluzione

(Funzioni)

```
float ipotenusa(float a, float b)
{
    return sqrt(a*a + b*b);
}
```

ESERCIZIO 8

(Funzioni)

Codificare in C la funzione

`float perimetro(float a, float b, float c)` che, dati i lati a,b,c di un triangolo, ne calcoli il perimetro.

Codificare in C la funzione

`float area(float a, float b, float c)`

che restituisca 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 prenda in ingresso i tre lati di un triangolo e stampi perimetro ed area.

ESERCIZIO 8 - Soluzione

(Funzioni)

```
float perimetro(float a, float b, float c)
{
    return a + b + c;
}
```

```
float area(float a, float b, float c)
{
    float p;
    float result;

    p = perimetro(a, b, c) / 2;
    result = sqrt(    p * (p-a) * (p-b) * (p-c)    );
    return result;
}
```

ESERCIZIO 9

(Funzioni)

Codificare in C la funzione `int isPrimo(int x)` che restituisca:

1 se x è un numero primo

0 altrimenti.

Si utilizzi a tal proposito l'operatore modulo (%).

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

ESERCIZIO 9 – Soluzione1

(Funzioni)

```
int isPrimo(int x) {
    int i, resto = 0;

    if (x == 1)
        return 0;
    else {
        if (x == 2)
            return 1;
        else {
            i = 2;
            do {
                resto = x % i;
                i++;
            } while ((resto != 0) && (i<x));
            return (resto != 0);
        }
    }
}
```


ESERCIZIO 9 – Soluzione2

(Funzioni)

```
int isPrimo(int x) {
    int i, resto, result;
    result = 0;

    if (x == 1)
        result = 0;
    else {
        if (x == 2)
            result = 1;
        else {
            i = 2;
            do {
                resto = x % i;
                i++;
            } while ((resto != 0) && (i<x));
            result = (resto != 0);
        }
    }
    return result;
}
```

ESERCIZIO 10

(Funzioni)

Codificare in C un programma che chieda di inserire un limite (un intero positivo), e stampi a video tutte le terne pitagoriche i cui valori sono compresi tra 1 e il limite indicato.

ESERCIZIO 10 – Soluzione

(Funzioni)

```
#include <stdio.h>

int mioMin(int a, int b) {
    if (a<b) return a;
    else return b;
}

int isTerna(int a, int b, int c) {
    int cateto1, cateto2, ipotenusa;
    int prod;

    cateto1 = mioMin(a,b);
    cateto2 = mioMin(a+b-cateto1, c);
    ipotenusa = a+b+c-cateto1-cateto2;
    prod = cateto1*cateto1+cateto2*cateto2;

    if (prod == ipotenusa*ipotenusa)
        return 1;
    else
        return 0;
}
```

ESERCIZIO 10 – Soluzione

(Funzioni)

```
int main() {
    int i, j, k;
    int limit;

    printf("Inserisci valore limite: ");
    scanf("%d", &limit);
    for (i=1; i<=limit; i++) {
        for (j=1; j<=i; j++) {
            for (k=1; k<=j; k++) {
                if (isTerna(i, j, k))
                    printf("Terna: %d %d %d\n", i, j, k);
            }
        }
    }

    return 0;
}
```

Nota: se la condizione del for fosse, ad esempio, $j \leq \text{limit}$, il programma stamperebbe triple ripetute.

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);  
}
```

```
// cosa succede se x<0 ???????
```

```
// come modificare la soluzione di cui sopra?
```

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 2 - Soluzione

(Funzioni ricorsive)

```
double f(double a, int n)
{ if (n==1) return a - 1/a;
  else return a - n/a + f(a, n-1);
}
```

```
double f(double a, int n)
{ int i=1;
  double sum=0;
  while(i<=n)
    {sum = sum + a - i/a;
     i++;}
  return sum;
}
```


Esercizio 3

(Funzioni ricorsive)

Si scriva un 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 3 - Soluzione

(Funzioni ricorsive)

```
int reverse(int num) {  
    return reverseRic(num, 0);  
}
```

```
int reverseRic(int num, int part) {  
    if (num == 0)  
        return part;  
    else {  
        return reverseRic(num/10, part*10 + num%10);  
    }  
}
```

```
int reverseIt(int num) {  
    int result = 0;  
    while (num!=0) {  
        result = result*10 + num%10;  
        num = num/10;  
    }  
    return result;  
}
```

Esercizio 4

(Funzioni ricorsive)

Si scriva un 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”

Esercizio 4 - Soluzione parziale

(Funzioni ricorsive)

```
void reverseChars() {
    char c;

    c = getchar();
    if (c == '\n')
        return;
    else {
        reverseChars();
        printf("%c", c);
        return;
    }
}
```

Esercizio 4 - Soluzione

(Funzioni ricorsive)

```
int reverseChars() {
    char c;
    int partialResult;

    c = getchar();
    if (c == '\n')
        return 0;
    else {
        partialResult = reverseChars();
        printf("%c", c);
        return partialResult + 1;
    }
}
```

Programmi su più moduli - Esempio

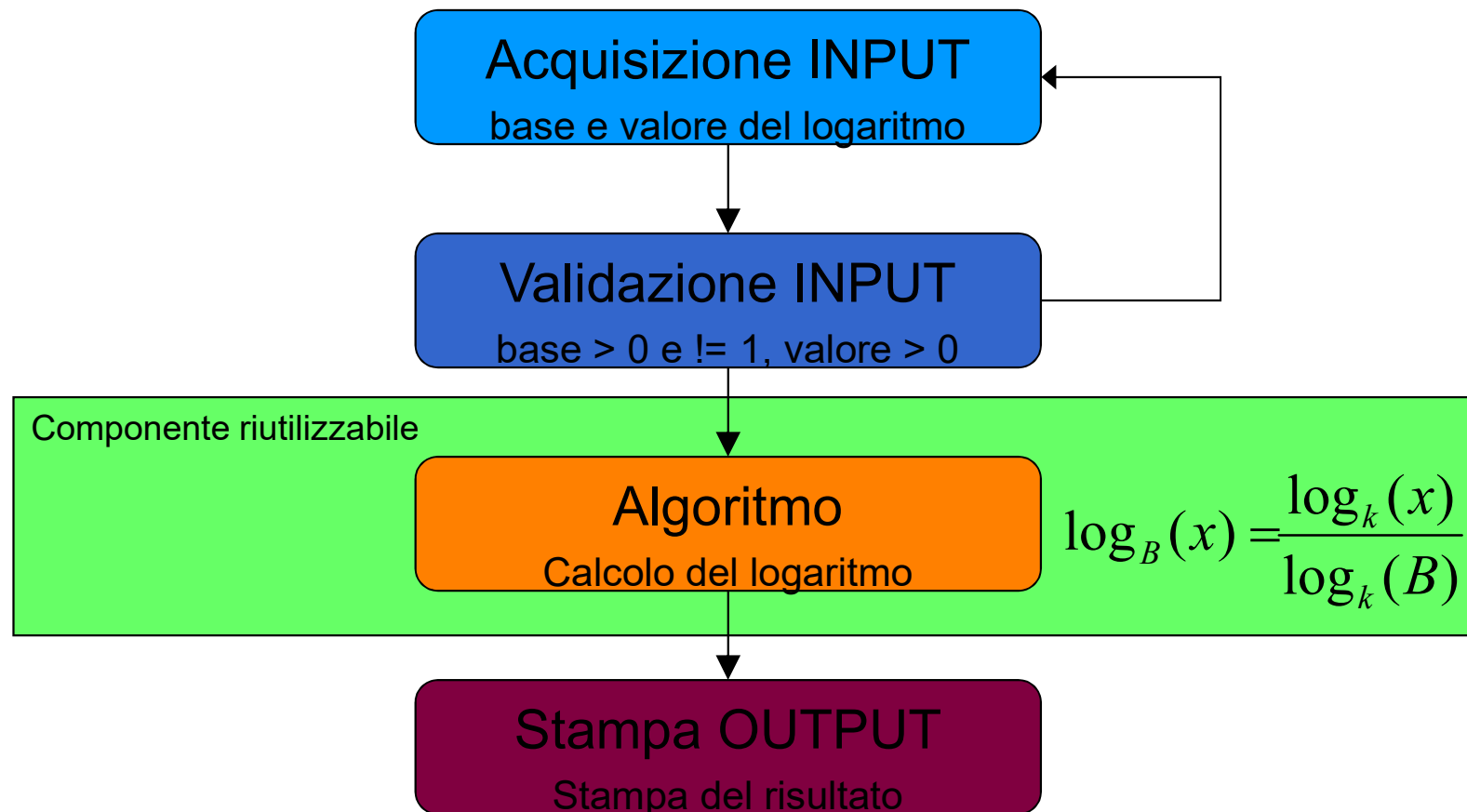
Calcolo del logaritmo in base qualunque

- 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

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

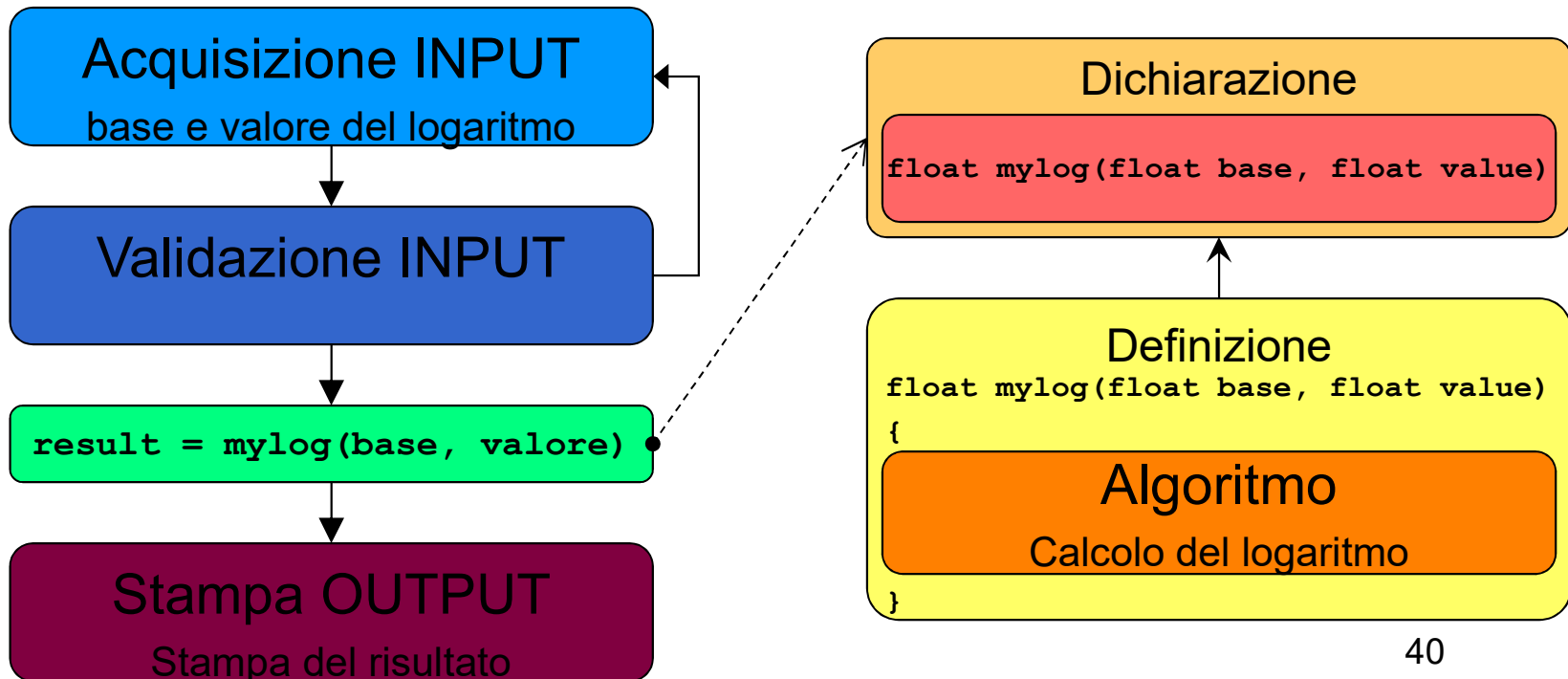
Programmi su più moduli - Esempio

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)

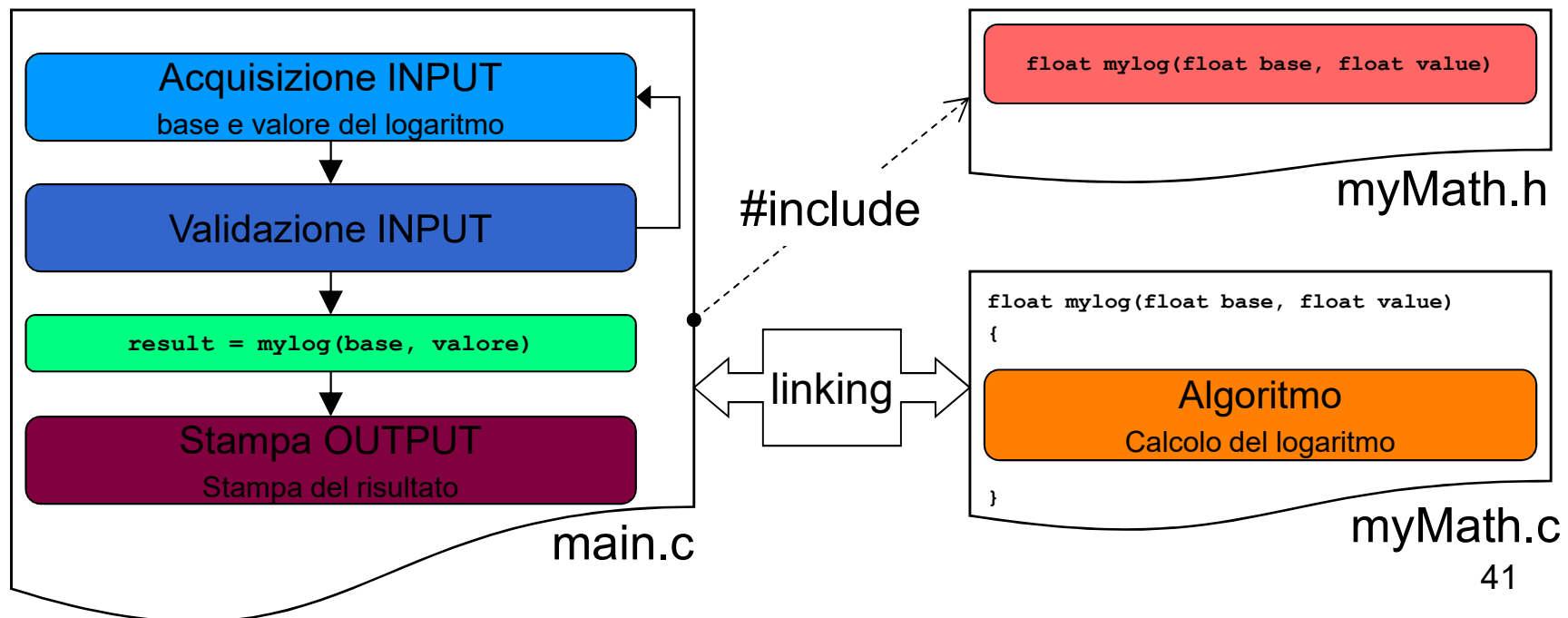


Programmi su più moduli - Esempio

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>
#include "myMath.h"
```

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

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

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 1 - Soluzione

(Funzioni e programmi su più moduli)

File “myMath.h”:

```
int miomax(int v1, int v2);  
int miomin(int v1, int v2);
```

File “myMath.c”:

```
#include "myMath.h"  
  
int miomax(int v1, int v2) {  
    if(v1 > v2)  
        return v1;  
    else  
        return v2;  
}  
  
int miomin(int v1, int v2) {  
    return v1 < v2 ? v1 : v2;  
}
```

Esercizio 1 - Soluzione

(Funzioni e programmi su più moduli)

File "main.c":

```
#include <stdio.h>
#include "myMath.h"
#define MAX_REQUEST 5

int main () {
    int curValue, maxValue, minValue, index;
    do {
        for(index = 0; index < MAX_REQUEST; index++) {
            printf("Inserire il valore %d: ", index+1);
            scanf("%d", &curValue);
            if(index == 0) //inizializzo max e min
            {
                maxValue = curValue;
                minValue = curValue;
            }
            else {
                maxValue = miomax(maxValue, curValue);
                minValue = miomin(minValue, curValue);
            }
        }
        printf("Calcolati: max = %d, min = %d\n", maxValue, minValue);
    } while(maxValue - minValue <= 10);
    return 0; }
```

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

$$\text{mcm}(a, b, c) = \text{mcm}(\text{mcm}(a, b), c)$$

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 $M \neq N$:
 - 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 dichiarazioni 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 2 - Soluzione

(Funzioni e programmi su più moduli)

File "myMath.h":

```
int mcd(int a, int b);  
int mcm(int a, int b);
```

File "myMath.c":

```
#include "myMath.h"  
int mcd(int a, int b) {  
    int m, n;  
    m = a;  
    n = b;  
    while(m != n) {  
        if(m > n)  
            m = m - n;  
        else  
            n = n - m;  
    }  
    return m;  
}  
int mcm(int a, int b) {  
    return (a * b) / mcd(a, b);  
}
```

Esercizio 2 - Soluzione

(Funzioni e programmi su più moduli)

File "main.c":

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

int main() {
    int curMcm, curValue;
    printf("Inserisci il primo valore: ");
    scanf("%d", &curMcm);
    do {
        printf("Inserisci un valore: ");
        scanf("%d", &curValue);
        curMcm = mcm(curMcm, curValue);
        printf("mcm corrente: %d\n", curMcm);
    } while(curMcm <= 100);
    return 0;
}
```

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

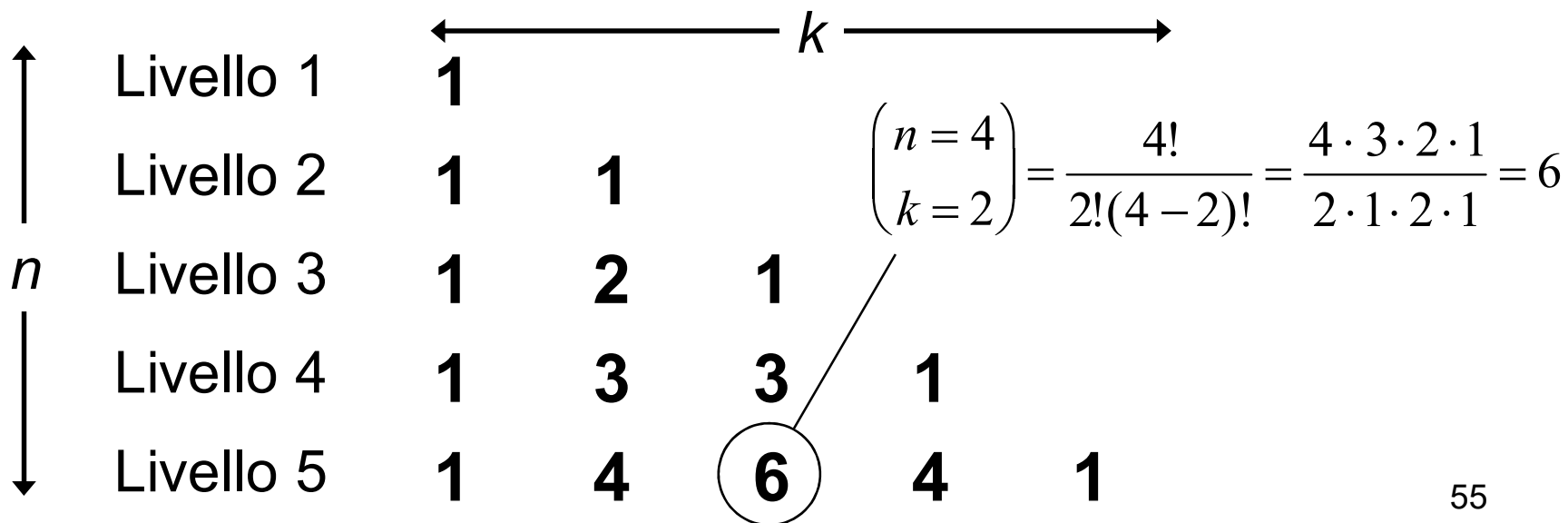
Esercizio 3

(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 $k \leq n$

Esercizio 3 - Soluzione

(Funzioni e programmi su più moduli)

File "myMath.h":

```
int fattoriale(int n);  
int binomiale(int n, int k)
```

File "myMath.c":

```
#include "myMath.h"  
int fattoriale(int n) {  
    int fact = 1, index;  
    for(index = n; index > 0; index--)  
        fact = fact * index;  
    return fact;  
}  
  
int binomiale(int n, int k) {  
    return fattoriale(n) / (fattoriale(k)*fattoriale(n-k));  
}
```

Esercizio 3 - Soluzione

(Funzioni e programmi su più moduli)

File "myMath.c":

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

int main () {
    int N_livelli, n, k;
    scanf("%d",&N_livelli);
    for(n = 0; n < N_livelli; n++) {
        for(k = 0; k <= n; k++)
            printf("%d ", binomiale(n, k) );
        printf("\n");
    }
    return 0;
}
```