



ELEMENTI DI INFORMATICA

DOCENTE: FRANCESCO MARRA

INGEGNERIA CHIMICA

INGEGNERIA ELETTRICA

SCIENZE ED INGEGNERIA DEI MATERIALI

INGEGNERIA GESTIONALE DELLA LOGISTICA E DELLA PRODUZIONE

INGEGNERIA NAVALE


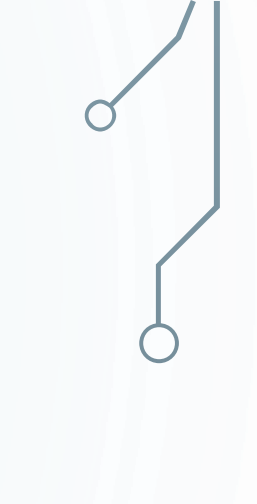
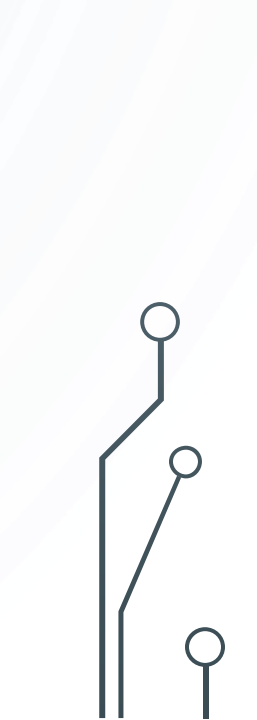
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

FUNZIONI E VISIBILITÀ





AGENDA

- Le funzioni
 - Definizione e valore di ritorno
 - Passaggio di parametri: valore e riferimento
 - Visibilità
- 
- 
- 

LE FUNZIONI



MODULARITÀ

- **Modularità**

- Uno degli aspetti più importanti della progettazione di un programma
- Suddivisione del programma in parti ben definite (sottoprogrammi) e codifica di ciascuna di esse come un modulo

MODULARITÀ

- Vantaggi
 - Ci si può concentrare su singoli aspetti della soluzione
 - Lavoro suddivisibile tra più persone
 - Semplificazione della dimostrazione della correttezza del programma
 - Riutilizzo dei moduli in altri progetti
 - Maggiore manutenibilità
 - Aumento della comprensione del programma

MODULARITÀ

- **Modulo**

- Blocco di istruzioni a cui viene assegnato un nome, scelto in base al compito assegnato al modulo
- `main()` è il modulo principale che può richiamare tutti gli altri moduli
- I linguaggi C e C++ applicano alla definizione di modulo il significato di *funzione*

DEFINIZIONE DI UNA FUNZIONE

- **Intestazione**

```
<type> funzione(tipo1 Pf1, tipo2 Pf2, ...)
```

- Tipo di risultato
- Nome della funzione
- Elenco dei tipi e nomi dei parametri formali

- **Corpo**

- Istruzioni racchiuse tra parentesi graffe
- L'ultima istruzione può restituire il risultato:
return

```
{  
    ...  
    ...  
    return risultato;  
}
```


ATTIVAZIONE DI UNA FUNZIONE

- Avviene attraverso il *richiamo*
 - del nome assegnato alla funzione
 - seguito dai parametri attuali (senza indicazione del tipo)
- Il richiamo diviene una nuova istruzione che produce come effetto l'esecuzione della funzione

```
int main()  
{  
    a=funzione(Pa1,Pa2, ...)  
}
```

RISULTATO

- Il tipo di risultato deve essere specificato nell'intestazione della funzione

```
<type> funzione(...)
```

- Il risultato da restituire è indicato con «return»

```
return <risultato>
```

- Quando si richiama una funzione tramite il suo nome, il risultato può essere inserito in espressioni che usano il valore restituito

```
a = funzione(...)
```

Il tipo di *risultato* deve essere compatibile con quello indicato nell'intestazione della funzione

PARAMETRI

- È possibile definire funzioni senza parametri (o argomenti)

`<type> funzione()`

Tipo restituito dalla funzione

Nome della funzione

Nessun argomento

PARAMETRI

- Un modulo è scritto in forma generalizzata usando variabili dette **Parametri formali**

```
<type> funzione(tipo1 Pf1, tipo2 Pf2, ...)
```

Es.:

```
float quadrato(float x_)
```

- All'attivazione del modulo, vengono associati ai parametri i dati effettivi sui quali il modulo deve effettuare i propri calcoli, detti **Parametri attuali** o **effettivi**

```
risultato = funzione(Pa1, Pa2, ...)
```

Es.:

```
q=quadrato(5);
```

Es.:

```
float x=5;  
q=quadrato(x);
```

Es.:

```
float v[10];  
v[0]=5;  
q=quadrato(v[0]);
```

PARAMETRI

- La lista dei parametri effettivi deve corrispondere in numero e tipo a quella dei parametri formali

```
void equazione_secondo_grado(float a, float b, float c, float &x1, float &x2, &bool radici_reali)
```

```
equazione_secondo_grado(c1, c2, c3, x1, x2, esist);
```



ESEMPIO: RESTITUZIONE RISULTATO

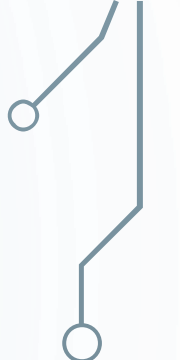
- Calcolo dell'ipotenusa di un triangolo mediante applicazione del teorema di Pitagora

```
float pitagora(float cateto_a, float cateto_b) {  
    float ipotenusa;  
    ipotenusa = sqrt(quadrato(cateto_a) + quadrato(cateto_b));  
    return ipotenusa;  
}
```

```
float pitagora(float cateto_a, float cateto_b) {  
    return sqrt(quadrato(cateto_a) + quadrato(cateto_b));  
}
```



ESERCIZIO 1

- Calcolare il quadrato di un numero immesso da tastiera
 - tramite una funzione che calcola il quadrato
- 

ASSOCIAZIONE DEI PARAMETRI EFFETTIVI AI FORMALI

- Il sottoprogramma

- necessita di valori iniziali: Parametri di ingresso
- produce risultati
 - Tramite il return si può produrre solo un risultato
 - Per produrre più risultati, si usano i Parametri di uscita

- Il programma chiamante

- deve fornire i valori dei parametri di ingresso
- deve ottenere
 - un risultato, oppure i valori dei parametri di uscita

```
void equazione_secondo_grado(  
    float a, float b, float c,  
    float &x1, float &x2,  
    bool &radici_reali )
```

Parametri di ingresso

Parametri di uscita

ASSOCIAZIONE DEI PARAMETRI EFFETTIVI AI FORMALI

- Sostituzione per valore
 - Il parametro effettivo viene ricopiato nel parametro formale all'esecuzione del sottoprogramma
 - Il sottoprogramma opera su una copia
 - **Il modulo non modifica il parametro effettivo**
- Sostituzione per riferimento
 - Viene fornito al parametro formale l'indirizzo di memoria del parametro effettivo
 - Il sottoprogramma opera direttamente sul parametro effettivo
 - **Il modulo può modificare il parametro effettivo**

ASSOCIAZIONE DEI PARAMETRI EFFETTIVI AI FORMALI

- Sostituzione per valore
 - Si usa solo la dichiarazione di tipo

```
void sottoprogramma(int num1, float num2, ...)
```

- Sostituzione per riferimento
 - Dichiarazione di tipo con operatore & **(solo con compilatori C++)**

```
void sottoprogramma(int &num1, float &num2, ...)
```

ASSOCIAZIONE DEI PARAMETRI EFFETTIVI AI FORMALI

- Sostituzione per valore
 - Usarla quando i parametri formali sono solo di ingresso al sottoprogramma
- Sostituzione per riferimento
 - Usarla quando i parametri formali sono
 - di uscita
 - sia di ingresso sia di uscita

PROCEDURE


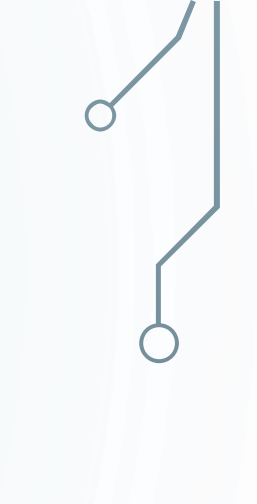
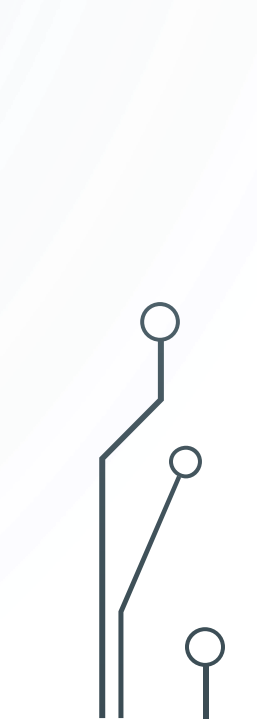
- Sono particolari funzioni che si usano nel caso di moduli che non producono un singolo risultato
 - Non deve essere usata l'istruzione *return*
 - Al tipo della funzione si deve attribuire il tipo *void*
 - La chiamata al modulo non può essere inserita in espressioni

```
void stampa_dato(int v) {  
    cout << "\nIl valore della variabile e' uguale a: ";  
    cout << v;  
}  
  
stampa_dato(5);
```

- In caso di moduli con parametri sostituiti per riferimento, si deve usare sempre il tipo *void*

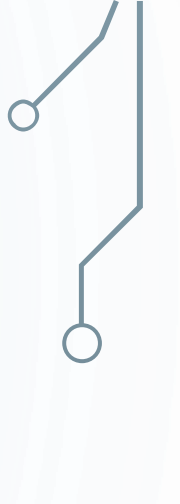


ESERCIZIO 2

- Realizzare un programma che consente di scambiare due valori interi inseriti da tastiera
 - Mediante la realizzazione di un sottoprogramma per eseguire lo scambio dei valori
- 
- 
- 



ESERCIZIO 3

- Realizzare un programma che calcola il massimo tra due numeri interi inseriti da tastiera
 - Mediante la realizzazione di un sottoprogramma che calcoli il massimo
- 

PARAMETRI FORMALI DI TIPO STRUTTURATO

- Il linguaggio prevede che possano essere passati ad una funzione variabili di tipo strutturato
- Nel caso di array, deve essere sempre usata la sostituzione per riferimento
 - Il carattere & è omesso

```
// Prototipi delle funzioni: entrambi corretti  
void calcola1(float v[100], int n); // dichiarazione completa  
void calcola2(float v[], int n);    // dichiarazione incompleta  
  
float vet[100];  
int n = 100;  
calcola1(vet,n);  
calcola2(vet,n);
```

ESEMPIO: PARAMETRI DI TIPO ARRAY

- Ordinamento in senso crescente dei valori di un vettore
 - Algoritmo detto *bubble sort*

```
void ordina(int vett[], int riemp) {  
    for(int i=0; i<riemp; i++)  
        for(int j=i+1; j<riemp; j++)  
            if(vett[i]>vett[j])  
                scambia(vett[i],vett[j]);  
}
```


PARAMETRI DI INGRESSO DI TIPO STRUTTURATO

- Per indicare che un parametro di tipo array è di ingresso, esso deve essere dichiarato *costante*
 - Si usa la parola chiave *const*

```
void calcola(const float v[], int n)
```

Parametro di ingresso di tipo vettore
di reali sostituito per riferimento

Parametro di ingresso di tipo
intero sostituito per valore

PASSAGGIO DI MATRICI

- Per le matrici valgono considerazioni analoghe
 - L'unica differenza è che la funzione deve essere informata del numero massimo di colonne della matrice

```
// Matrice m con numero di righe imprecisate e 50 colonne  
void calcola(float m[][50]);  
  
float matr[10][50];  
calcola(matr);
```

ESEMPIO: PARAMETRO DI TIPO MATRICE

- Funzione che determina il numero più grande in una matrice

```
float max_mat(float mat[][100], int n, int m) {  
    float max = mat[0][0];  
    for(int i=0; i<n; i++) {  
        for(int j=0; j<m; j++) {  
            if(max < mat[i][j]) {  
                max = mat[i][j];  
            }  
        }  
    }  
    return max;  
}
```

PASSAGGIO DI ARRAY MULTIDIMENSIONALI

- Il caso degli array a più di due dimensioni è una generalizzazione del caso della matrice
 - Nella dichiarazione del parametro formale l'unica cardinalità che può non essere espressa è la prima

```
void calcola(float m[][10][20][5])
```

USO DI TYPEDEF CON ARRAY

- La corrispondenza tra parametri effettivi e formali può essere agevolata dall'utilizzo di *typedef*
 - Semplifica la scrittura di programmi e limita l'introduzione di errori

```
typedef int tipo_matrice[50][100];

float max_mat(const tipo_matrice mat, int n, int m) {
    ...
}

int main() {
    tipo_matrice matrice;
    int riemp_r, riemp_c;
    float x;
    ...
    x = max_mat(matrice, riemp_r, riemp_c);
    ...
}
```

USO DI TYPEDEF CON RECORD

```
typedef struct {  
    char nome[30];  
    char cognome[30];  
    int eta;  
} t_anagrafico;  
  
void cerca(const t_anagrafico an, int numero, bool &trovato) {  
    ...  
}  
  
int main() {  
    t_anagrafico anagrafico;  
    bool ok;  
    ...  
    cerca(anagrafico, 100, ok);  
    ...  
}
```

TIPI DI FUNZIONI: DIFFERENZE

■ Funzione con return:

■ Tipo:

- int
- float
- bool
- ...

■ Parametri:

- Sostituiti per valore

■ Parametri di uscita:

- Solo 1
- .

• Procedura:

• Tipo:

- void

• Parametri:

- Sostituiti per valore
- Sostituiti per riferimento
- Entrambi

• Parametri di uscita:

- Nessuno
- 1 o più
- 1 o più

ESERCIZIO 4

- Calcolo del minimo, del massimo e della media di un qualsivoglia numero di valori immessi da tastiera
 - Tramite una funzione che calcola minimo, massimo e media di un vettore

VISIBILITÀ DI UN IDENTIFICATORE

La visibilità è la porzione di programma dove l'identificatore si può usare.

Valgono le seguenti regole:

- Un identificatore per essere usato deve essere **dichiarato**
- Un identificatore è visibile solo dalle istruzioni che **seguono** la sua definizione
- Se la dichiarazione è interna ad un blocco, l'identificatore è visibile **solo all'interno** del blocco
- Le variabili dichiarate in un blocco sono dette **variabili locali**
- Se la dichiarazione è esterna ad un blocco, l'identificatore è visibile **anche all'interno** del blocco
- Le variabili dichiarate all'esterno di tutti i blocchi di un programma sono dette **variabili globali**
- È possibile introdurre un identificatore *già dichiarato* esternamente
- All'interno del blocco viene considerato diverso dagli altri

ESEMPIO: VISIBILITÀ IDENTIFICATORI

```
#include <iostream>

using namespace std;

int main() {
    int a = 7;
    float b = 15.5;

    for(int i=0; i<10; i++) {
        cout << "Ciclo n. ";
        int a = i+1;
        cout << a << endl;
    }

    cout << "Il valore di a e': " << a << endl;
    system("PAUSE");
    return 0;
}
```

The diagram illustrates the visibility of variables 'a', 'b', and 'i' across different scopes in the provided C++ code. Colored boxes represent the scope of each variable:

- Red box (labeled 'a'):** Represents the scope of the variable 'a' declared inside the for loop. It covers the lines from `cout << "Ciclo n. ";` to `cout << a << endl;`.
- Blue box (labeled 'b'):** Represents the scope of the variable 'b' declared in the `main` function. It covers the entire `main` function body, from `int a = 7;` to `return 0;`.
- Green box (labeled 'i'):** Represents the scope of the variable 'i' declared in the for loop. It covers the entire for loop body, from `for(int i=0; i<10; i++) {` to `}`.

The diagram shows that the variable 'a' is only visible within the for loop, while 'b' is visible throughout the `main` function. The variable 'i' is only visible within the for loop.

VARIABILI LOCALI

- Tutto ciò che è definito all'interno di un sottoprogramma è **locale** ad esso
 - Variabili
 - Parametri formali
- La località degli identificatori garantisce il disaccoppiamento tra programma chiamante e sottoprogramma
- Il valore di una variabile locale può essere usato solo dalla funzione in cui è dichiarata
- Nulla può dirsi sul valore assunto all'avvio della funzione, in quanto vengono deallocate dalla memoria quando la funzione termina

```
void equazione_secondo_grado(float a, float b, float c,
                             float &x1, float &x2,
                             bool&radici_reali) {

    float det;
    radici_reali=true;
    det = b*b - 4*a*c;
    if (det>0) { // radici reali e distinte
        det = sqrt(det);
        x1 = (-b-det) / (2*a);
        x2 = (-b+det) / (2*a);
    }
    else if (det==0) { // radici coincidenti
        x1 = x2 = -b / (2*a);
    }
    else { // radici immaginarie
        radici_reali=false;
    }
} // fine modulo equazione_secondo_grado
```

```
int main()
{
    float x=0, y=0;
    bool risultatoReale;
    equazione_secondo_grado(1,0,-4,x,y,risultatoReale);

    cout << det;
    cout << b;
```

	C:\Dev-Cpp\main.cpp	In function 'int main()':
31	C:\Dev-Cpp\main.cpp	'det' undeclared (first use this function) (Each undeclared identifier is reported only once for each function it appears in.)
32	C:\Dev-Cpp\main.cpp	'b' undeclared (first use this function)

VARIABILI NON LOCALI

- L'utilizzo di variabili globali in una funzione può comportare *effetti collaterali*
- La modifica in una funzione di una variabile non locale potrebbe ripercuotersi anche *esternamente*
- Nell'esempio, il ciclo *for* interno al *main* termina dopo una sola iterazione

```
#include <iostream>

using namespace std;

int i; // variabile globale

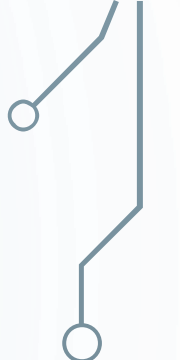
int potenza(int x, int n) {
    int potenza = x;
    for(i=1; i<n; i++) {
        potenza *= x;
    } // il for termina con i pari a n
    return potenza;
}

int main() {
    int n, p;

    cout << "Inserisci n: ";
    cin >> n;
    for(i=1; i<=n; i++) {
        cout << "La potenza di " << i << " alla " << n;
        p = potenza(i,n);
        cout << "e' " << p;
    }
    return 0;
}
```



CICLO DI VITA

- *Ciclo di vita* di una variabile
 - Intervallo compreso tra la sua creazione e la sua eliminazione
 - In generale inizia con la dichiarazione e termina con la chiusura del blocco
- 

ESEMPIO: VISIBILITÀ VARIABILI

```
int v1, v2;

float funzione_a(int x) {
    float v3;
}

float funzione_b(char c) {
    int v1, v4;
}

double v3;

void funzione_c() {
    int v5;
}

int main() {
    double v1, v5;
    ...
}
```


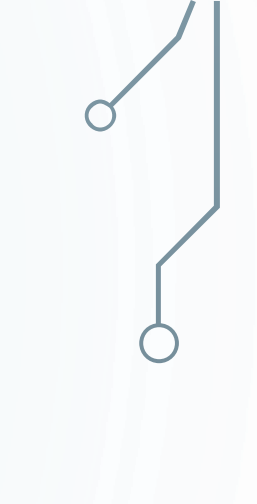
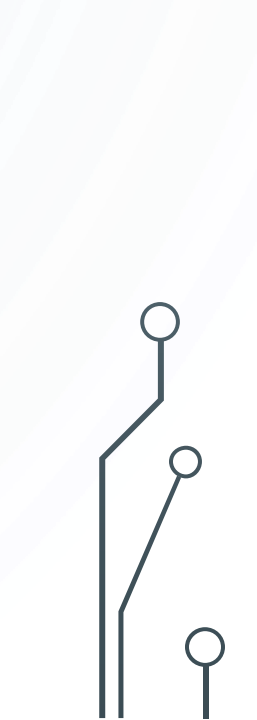
Modulo	Vede modulo	Variabili non locali	Variabili locali
funzione_a	funzione_a	v1, v2	x, v3
funzione_b	funzione_a funzione_b	v2	c, v1, v4
funzione_c	funzione_a funzione_b funzione_c	v1, v2, v3	v5
main	funzione_a funzione_b funzione_c	v2, v3	v1, v5

VISIBILITÀ FUNZIONI





STRUTTURA DI UN PROGRAMMA

- L'esecuzione di un programma inizia con la prima istruzione del *main*
 - Si prosegue con le altre istruzioni del *main*
 - Possono essere chiamate altre funzioni che ne richiamano altre a loro volta
- 
- 
- 

DICHIARAZIONI DI FUNZIONI

- Tutte le funzioni sono dichiarate all'inizio del programma
 - Si rispettano le regole di visibilità
- Il *main* segue tutte le dichiarazioni di funzione
- Non è possibile dichiarare una funzione all'interno di un'altra
 - Appiattimento su di uno stesso livello

DICHIARAZIONI DI FUNZIONI

- Non è sempre possibile far precedere la dichiarazione di funzione al suo richiamo!

```
void funzione_a(int x)
{
    .....
    funzione_c();
}

void funzione_b(char c)
{
    .....
    funzione_a(k);
}

void funzione_c()
{
    .....
    funzione_b(x);
    funzione_d(y);
}
```

```
void funzione_d(char c)
{
    .....
    funzione_c();
    funzione_b(z);
}

int main()
{
    .....
    funzione_a(i);
}
```

PROTOTIPO DI FUNZIONE

- Versione semplificata dell'intestazione di funzione terminata da punto e virgola

```
<Type> nome_funzione(elenco dei tipi dei parametri);
```

- Fornisce al compilatore l'elenco delle funzioni da richiamare
 - Indica la struttura della funzione
 - Tipo di ritorno
 - Nome
 - Tipo e nome degli argomenti, se presenti
 - Il nome degli argomenti è facoltativo

PROTOTIPO DI FUNZIONE: ESEMPIO

```
float integrale(float estremo_a, float estremo_b, float& precisione)
```



```
float integrale(float, float, float&);
```

Oppure

```
float integrale(float estremo_a, float estremo_b, float& precisione);
```

STRUTTURA DI PROGRAMMA CON PROTOTIPI

```
/* programma per il calcolo
dell'ipotenusa di un
triangolo */

#include <iostream>
#include <math>
using namespace std;

/* elenco prototipi delle funzioni */
float quadrato(float);
float pitagora(float, float);

int main (){
    float lato_a, lato_b;

    .....

    pitagora(lato_a, lato_b);

    .....

}
```

L'ordine di scrittura di prototipi e dichiarazioni complete è influente

In testa al *main* vengono elencati i prototipi delle funzioni da richiamare

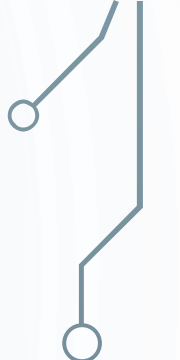
```
/* dichiarazione delle funzioni */
float quadrato(float x) {
    return x*x;
}

float pitagora(float lato_a, float lato_b){
    return sqrt(quadrato(lato_a)
                + quadrato(lato_b));
}
```

In coda al *main* vengono riportate le dichiarazioni complete delle funzioni



ESERCIZIO

- Realizzare un programma che determina se un numero è presente in un vettore e che ne restituisca la posizione in caso affermativo
 - Mediante la realizzazione di una funzione
- 

DOMANDE, DUBBI, PERPLESSITÀ

