

Esercitazione 08

2026/10/23 – Funzioni

L. Alessandrini

Credits: A. Montenegro per gli esercizi

Funzioni

Recap

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

Funzioni

Caratteristiche Generali

Tipo della variabile restituita

```
int somma(int a, int b){  
    return a+b;  
}
```

Funzioni

Caratteristiche Generali

Tipo della variabile restituita

```
int somma(int a, int b){  
    return a+b;  
}
```

Può anche essere:

- Puntatore

Funzioni

Caratteristiche Generali

Tipo della variabile restituita

```
int somma(int a, int b){  
    return a+b;  
}
```

Può anche essere:

- Puntatore
- Nuovo tipo definito

Funzioni

Caratteristiche Generali

Tipo della variabile restituita

```
int somma(int a, int b){  
    return a+b;  
}
```

Può anche essere:

- Puntatore
- Nuovo tipo definito
- void se la funzione non restituisce nulla

Funzioni

Caratteristiche Generali

Nome della Funzione

```
int somma(int a, int b){  
    return a+b;  
}
```

Funzioni

Caratteristiche Generali

Argomenti

```
int somma(int a, int b){  
    return a+b;  
}
```

Funzioni

Caratteristiche Generali

Argomenti

```
int somma(int a, int b){  
    return a+b;  
}
```

Caratteristiche:

- Specifica sia il tipo che il nome!

Funzioni

Caratteristiche Generali

Argomenti

```
int somma(int a, int b) {  
    return a+b;  
}
```

Caratteristiche:

- Specifica sia il tipo che il nome!
- Possono essere usati all'interno del corpo della funzione

Funzioni

Caratteristiche Generali

Argomenti

```
int somma(int a, int b) {  
    return a+b;  
}
```

Caratteristiche:

- Specifica sia il tipo che il nome!
- Possono essere usati all'interno del corpo della funzione
- Sono passati per **copia**

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

Corpo della funzione

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

Corpo della funzione

Caratteristiche:

Qui si esegue il codice della funzione.
Nota: si possono usare le variabili in input!

Funzioni

Caratteristiche Generali

- Tutte le variabili che vengono create nella funzione vengono distrutte quando essa termina

```
int somma(int a, int b){  
    return a+b;  
}
```

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

- Tutte le variabili che vengono create nella funzione vengono distrutte quando essa termina
- La funzione NON vede le variabili delle altre funzioni (e quindi anche del main)

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

- Tutte le variabili che vengono create nella funzione vengono distrutte quando essa termina
- La funzione NON vede le variabili delle altre funzioni (e quindi anche del main)
- La funzione vede tutto ciò che è fuori dalle altre funzioni:

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

- Tutte le variabili che vengono create nella funzione vengono distrutte quando essa termina
- La funzione NON vede le variabili delle altre funzioni (e quindi anche del main)
- La funzione vede tutto ciò che è fuori dalle altre funzioni:
 1. #define N ...

Funzioni

Caratteristiche Generali

```
int somma(int a, int b){  
    return a+b;  
}
```

- Tutte le variabili che vengono create nella funzione vengono distrutte quando essa termina
- La funzione NON vede le variabili delle altre funzioni (e quindi anche del main)
- La funzione vede tutto ciò che è fuori dalle altre funzioni:
 1. #define N ...
 2. Typedef ... (solo se non sono nel main)

Funzioni

Caratteristiche Generali

```
int main(){  
    int n1, n2, res;  
    n1 = 1;  
    n2 = 3;  
    res = somma(n1, n2);  
    printf("%d", res);  
}
```

Funzioni

Caratteristiche Generali

```
int main(){  
    int n1, n2, res;  
    n1 = 1;  
    n2 = 3;  
    res = somma(n1, n2);  
    printf("%d", res);  
}
```



```
int somma(int a, int b){  
    return a+b;  
}
```

The diagram illustrates a function call. A blue arrow originates from the `somma(n1, n2);` line in the `main` function and points to the `somma` function definition, which is enclosed in a grey box with a blue border.

Funzioni

Caratteristiche Generali

I valori di n1 e n2
vengono **copiati** negli
input della funzione

```
int main(){  
    int n1, n2, res;  
    n1 = 1;  
    n2 = 3;  
    res = somma(n1, n2);  
    printf("%d", res);  
}
```

```
int somma(int a, int b){  
    return a+b;  
}
```



Funzioni

Caratteristiche Generali

I valori di n1 e n2
vengono **copiati** negli
input della funzione

```
int main(){  
    int n1, n2, res;  
    n1 = 1;  
    n2 = 3;  
    res = somma(n1, n2);  
    printf("%d", res);  
}
```

```
int somma(int a, int b){  
    return a+b;  
}
```



Se si modificano a e b nel corpo della funzione, non si ha
nessun effetto su n1 e n2

Funzioni

Caratteristiche Generali

Per avere un effetto del genere, bisogna passare i **puntatori alle variabili**

input della funzione

```
int main(){  
    int n1, n2, res;  
    n1 = 1;  
    n2 = 3;  
    res = somma(n1, n2);  
    printf("%d", res);  
}
```

```
int somma(int a, int b){  
    return a+b;  
}
```

Se si modificano a e b nel corpo della funzione, non si ha **nessun effetto** su n1 e n2

Funzioni

Dichiarazione

Come si dichiarano le funzioni?

Ci sono due modi:

1. Con prototipo
2. Senza prototipo

Funzioni

Senza Prototipo

- Le funzioni vengono implementate direttamente
- Ogni funzione va scritta PRIMA di una funzione che ne fa uso

Funzioni

Senza Prototipo

- Le funzioni vengono implementate direttamente
- Ogni funzione va scritta PRIMA di una funzione che ne fa uso

```
#include <stdio.h>

int leggiNumero() {
    int a;
    scanf("%d", &a);
    return a;
}

int somma(int a, int b) {
    return a + b;
}

int main(){
    int n1, n2, res;
    n1 = leggiNumero();
    n2 = leggiNumero();
    res = somma(n1, n2);
    return 0;
}
```

Funzioni

Senza Prototipo

- Le funzioni vengono implementate direttamente
- Ogni funzione va scritta PRIMA di una funzione che ne fa uso



```
#include <stdio.h>

int leggiNumero() {
    int a;
    scanf("%d", &a);
    return a;
}

int somma(int a, int b) {
    return a + b;
}

int main(){
    int n1, n2, res;
    n1 = leggiNumero();
    n2 = leggiNumero();
    res = somma(n1, n2);
    return 0;
}
```

Funzioni

Con Prototipo

Il prototipo di una funzione è lo scheletro di essa. Bisogna definire solo:

- Tipo restituito
- Nome
- Tipi degli argomenti

NON bisogna definire:

- Nomi degli input
- Corpo della funzione

Funzioni

Con Prototipo



Funzione

```
int somma(int a, int b){  
    return a+b;  
}
```

```
int leggi(){  
    int a;  
    scanf("%d", &a);  
    return a;  
}
```

Prototipo

Funzioni

Con Prototipo



Funzione

```
int somma(int a, int b){  
    return a+b;  
}
```

Prototipo

```
int somma(int, int);
```

```
int leggi(){  
    int a;  
    scanf("%d", &a);  
    return a;  
}
```

Funzioni

Con Prototipo



Funzione

```
int somma(int a, int b){  
    return a+b;  
}
```

Prototipo

```
int somma(int, int);
```

```
int leggi(){  
    int a;  
    scanf("%d", &a);  
    return a;  
}
```

```
int leggi();
```

A blue arrow points from the function definition box on the left to the function prototype box on the right, indicating that the prototype is a declaration of the function defined below.

Funzioni

Con Prototipo

1. Si definiscono tutti i prototipi



```
#include <stdio.h>

int leggiNumero();
int somma(int, int);

int main() {
    int n1, n2, res;
    n1 = leggiNumero();
    n2 = leggiNumero();
    res = somma(n1, n2);
    return 0;
}

int leggiNumero() {
    int a;
    scanf("%d", &a);
    return a;
}

int somma(int a, int b) {
    return a + b;
}
```

Funzioni

Con Prototipo

1. Si definiscono tutti i prototipi
2. Poi si definiscono le funzioni



```
#include <stdio.h>

int leggiNumero();
int somma(int, int);

int main() {
    int n1, n2, res;
    n1 = leggiNumero();
    n2 = leggiNumero();
    res = somma(n1, n2);
    return 0;
}

int leggiNumero() {
    int a;
    scanf("%d", &a);
    return a;
}

int somma(int a, int b) {
    return a + b;
}
```

Funzioni

Con Prototipo

1. Si definiscono tutti i prototipi
2. Poi si definiscono le funzioni
 - Senza pensare all'ordine di implementazione



```
#include <stdio.h>

int leggiNumero();
int somma(int, int);

int main() {
    int n1, n2, res;
    n1 = leggiNumero();
    n2 = leggiNumero();
    res = somma(n1, n2);
    return 0;
}

int leggiNumero() {
    int a;
    scanf("%d", &a);
    return a;
}

int somma(int a, int b) {
    return a + b;
}
```

Funzioni

Con Prototipo



1. Si definiscono tutti i prototipi
2. Poi si definiscono le funzioni
 - Senza pensare all'ordine di implementazione
 - Tutto molto ordinato

```
#include <stdio.h>

int leggiNumero();
int somma(int, int);

int main() {
    int n1, n2, res;
    n1 = leggiNumero();
    n2 = leggiNumero();
    res = somma(n1, n2);
    return 0;
}

int leggiNumero() {
    int a;
    scanf("%d", &a);
    return a;
}

int somma(int a, int b) {
    return a + b;
}
```

Funzioni

Esercizi – Parte 1

Esercizio 00

Somma e Differenza

Testo

Scrivere un programma che chiede all'utente di inserire un numero naturale. Scrivere le funzioni:

- `leggiN()` : che legge da input e restituisce un numero naturale
- `somma()` : dati due interi, restituisce la somma dei due
- `diff()` : come la somma, ma per la differenza
- `sommaDiff()` : prende in input n_1 , n_2 , e restituisce sia $n_1 + n_2$ che $n_1 - n_2$

Nota! In generale quando si dice “prende in input” delle variabili, queste non sono TUTTE e SOLE le variabili in input, potrebbe esserci altro!

Esercizio 01

Divisori di un Numero

Testo

Scrivere un programma che chiede all'utente di inserire un numero naturale. Il programma stampa tutti i divisori del numero. Scrivere le funzioni:

- `leggiNumeroNaturale()` : che legge da input e restituisce un numero naturale
- `stampaDivisori()` : che dato un numero naturale, stampa tutti i suoi divisori

Esercizio 02

Numeri Amici

Testo

Scrivere un programma che:

1. prende in input due numeri naturali $n1$ e $n2$
2. verifica che i numeri siano amici
3. stampa se i numeri sono amici o meno

Scrivere le funzioni:

- `leggiNumeroNaturale()`: che legge e restituisce un numero naturale
- `sommaDivisori()`: che restituisce la somma di tutti i divisori di un numero dato in input
- `numeriAmici()`: che prende in input due numeri naturali e restituisce 1 quando i numeri sono amici, 0 altrimenti.

Nota: $n1$ e $n2$ sono amici se $n1$ è uguale alla somma dei divisori propri (i.e., eccetto se stesso) di $n2$ e viceversa.

Esercizio 02

Numeri Amici

Testo

Scrivere un programma che:

1. prende in input due numeri naturali $n1$ e $n2$
2. verifica che i numeri siano amici
3. stampa se i numeri sono amici o meno

Scrivere le funzioni:

- `leggiNumeroNaturale()`: che legge e restituisce un numero naturale
- `sommaDivisori()`: che restituisce la somma di tutti i divisori di un numero dato in input
- `numeriAmici()`: che prende in input due numeri naturali e restituisce 1 quando i numeri sono amici, 0 altrimenti.

Nota: $n1$ e $n2$ sono amici se $n1$ è uguale alla somma dei divisori propri (i.e., eccetto se stesso) di $n2$ e viceversa.

Nota! La funzione `sommaDivisori()` ritorna la somma di tutti i divisori ... Non solo i propri!

Esercizio 03

Numeri Perfetti

Testo

Scrivere un programma che chiede all'utente di inserire un numero naturale. Il programma verifica che il numero sia perfetto.

Nota: un numero si dice perfetto quando è uguale alla somma di tutti i suoi divisori (eccetto se stesso).

Scrivere le funzioni:

- `leggiNumeroNaturale()` : che legge da input e restituisce un numero naturale
- `sommaDivisori()` : che dato un numero naturale, restituisce la somma di tutti i suoi divisori
- `numeroPerfetto()` : che dato un numero naturale, restituisce 1 se il numero è perfetto, 0 altrimenti.

Esercizio 04

Numeri Perfetti, Abbondanti e Difettivi

Testo

Scrivere un programma che chiede all'utente di inserire un numero naturale. Il programma verifica che il numero sia perfetto, abbondante o difettivo.

Nota: un numero si dice perfetto quando è uguale alla somma di tutti i suoi divisori (eccetto se stesso). Il numero è abbondante se è minore della somma dei suoi divisori propri. Il numero è difettivo se è maggiore della somma dei suoi divisori propri.

Scrivere le funzioni:

- `leggiNumeroNaturale()` : che legge da input e restituisce un numero naturale
- `sommaDivisori()` : che dato un numero naturale, restituisce la somma di tutti i suoi divisori
- `numeroPerfettoPlus()` : che dato un numero naturale, restituisce 1 se il numero è perfetto, 0 altrimenti. Se il numero non è perfetto, la funzione deve essere in grado di comunicare al main se il numero è abbondante o difettivo.

Esercizio 05

Calcolo di Potenze

Testo

Scrivere un programma che calcola la potenza di un numero elevato a un esponente.

Il programma deve: chiedere due numero in input all'utente, uno sarà la base (anche negativo), l'altro l'esponente (da considerarsi solo positivo); calcolare la base elevata alla potenza.

Scrivere le funzioni:

- `leggiNum( )`: che legge e restituisce un numero intero
- `leggiNumPos( )`: che legge e restituisce un numero intero positivo
- `pow( )`: che prende in input base ed esponente e restituisce la base elevata all'esponente (base intera, esponente intero positivo)

Esercizio 06

Successione di Fibonacci

Testo

Scrivere un programma che:

1. chiede in input un numero naturale positivo
2. stampa la serie di fibonacci fino a quel numero

Scrivere le funzioni:

- `leggiNumeroPositivo()` :che legge e restituisce un intero positivo
- `stampaFibonacci()` : che prende in input un intero positivo e stampa la serie di fibonacci fino a qual punto

Esercizio 07

Calcolatrice 2.0

Testo

Scrivere un programma che, fin quando l'utente non ordina la terminazione inserendo 'q' o 'Q', permette di:

1. inserire due numeri
2. inserire un tipo di operazione: +, -, *, /
3. svolge e stampa a schermo il risultato dell'operazione

Scrivere le funzioni:

- `somma()`: presi due numeri ne restituisce la somma
- `differenza()`: presi due numeri ne restituisce la differenza
- `prodotto()`: presi due numeri ne restituisce il prodotto
- `divisione()`: presi due numeri ne restituisce la divisione e se il secondo operando è 0, comunica al chiamante che c'è stato un errore
- `leggiNumero()`: che legge e restituisce un numero lasciando il buffer pulito
- `leggiCarattere()`: che legge e restituisce un carattere lasciando il buffer pulito
- `menu()`: che implementa la logica della calcolatrice in un ciclo che dura fin quando l'utente lo vuole. Non ha nulla in input e non restituisce nulla

Esercizio 09

Rettangolo: Appartenenza, Perimetro e Area

Testo

Si definisca un tipo di dato che rappresenti un punto nel piano cartesiano. Si definisca un altro tipo di dato che rappresenti quattro vertici di un rettangolo nel piano cartesiano.

Si scriva un programma che:

1. chiede in input un punto
2. chiede in input un rettangolo
3. verifica se il punto si trova all'interno del rettangolo
4. calcola area e perimetro del rettangolo

Si scrivano le funzioni:

- `leggiPunto()`: che legge le coordinate di un punto e restituisce tale punto
- `leggiRettangolo()`: che legge i punti di un rettangolo e restituisce un rettangolo. Imporre dei controlli per verificare che i punti inseriti siano effettivamente parte di un rettangolo.
- `nelRettangolo()`: che dato un punto e un rettangolo, restituisce 1 se il punto è all'interno del rettangolo e 0 altrimenti
- `calcolaArea()`: che dato un rettangolo restituisce la sua area
- `calcolaPerimetro()`: che dato un rettangolo restituisce il suo perimetro
- `calcolaAeraPerimetro()`: che dato un rettangolo, non restituisce nulla, ma fa in modo che la funzione chiamante abbia visione del perimetro e dell'area del rettangolo

Thank You!

Mail: luca1.alessandrini@polimi.it

Website: <https://alessandriniluca.github.io/>