

# Funzioni

## Esercitazione 7

28/10/2024 - Alessandro Montenegro

# 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;  
}
```

Anche:

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

# Funzioni

## Caratteristiche Generali

Tipo della variabile  
restituita

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

Nome della funzione

# Funzioni

## Caratteristiche Generali

Tipo della variabile restituita

Nome della funzione

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

Argomenti

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

# Funzioni

## Caratteristiche Generali

Tipo della variabile restituita

Argomenti

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

Nome della funzione

Corpo

- Qui si scrive il codice eseguito dalla funzione
- Si possono usare le variabili passate in input

# 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)
- A meno che

# 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;  
}
```

# 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

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

```
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

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

```
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

## Dichiarazione

Ci sono due modi per definire le funzioni

- Con prototipo
- Senza prototipo

# Funzioni

## Dichiarazione Senza Prototipo



```
1  #include <stdio.h>
2
3  int leggiNumero(){
4      int a;
5      scanf("%d", &a);
6      return a;
7  }
8
9  int somma(int a, int b){
10     return a + b;
11 }
12
13 int main(){
14     int n1, n2, res;
15     n1 = leggiNumero();
16     n2 = leggiNumero();
17     res = somma(n1, n2);
18     return 0;
19 }
```

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

# Funzioni

## Dichiarazione 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

## Dichiarazione Con Prototipo



### Funzione

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

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

### Prototipo

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

```
int leggi();
```

# Funzioni

## Dichiarazione Con Prototipo



```
1  #include <stdio.h>
2
3  int leggiNumero();
4  int somma(int, int);
5
6  int main(){
7      int n1, n2, res;
8      n1 = leggiNumero();
9      n2 = leggiNumero();
10     res = somma(n1, n2);
11     return 0;
12 }
13
14 int leggiNumero(){
15     int a;
16     scanf("%d", &a);
17     return a;
18 }
19
20 int somma(int a, int b){
21     return a + b;
22 }
```

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

# Divisori di Un Numero

## Esercizio 1

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

# Numeri Amici

## Esercizio 2

Scrivere un programma che:

1. prende in input due numeri naturali  $n_1$  e  $n_2$
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:**  $n_1$  e  $n_2$  sono amici se  $n_1$  è uguale alla somma dei divisori propri (i.e., eccetto se stesso) di  $n_2$  e viceversa.

# Numeri Perfetti

## Esercizio 3

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.

# Numeri Perfetti, Abbondanti e Difettivi

## Esercizio 4

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.

# Calcolo di Potenze

## Esercizio 5

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

# Successione di Fibonacci

## Esercizio 6

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

# Calcolatrice v2.0

## Esercizio 7

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

# Rettangolo: Appartenenza, Perimetro e Area

## Esercizio 8

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

# Contatti

Alessandro Montenegro

Mail: [alessandro.montenegro@polimi.it](mailto:alessandro.montenegro@polimi.it)

Sito: <https://montenegroalessandro.github.io/InfoA2425/index.html>