



Fondamenti di Informatica
Ingegneria Clinica
Lezione 18/11/2010



Prof. Raffaele Nicolussi
FUB - Fondazione Ugo Bordoni
Via del Policlinico, 147 - 00161 Roma



Docente	Raffaele Nicolussi	<i>rnicolussi@fub.it</i> <i>0654803323</i>
Lezioni Aula 54 (ex aula 4) Via del Castro Laurenziano, 7	Lunedì, Giovedì, Venerdì	12:00 – 13:30
Esercitazioni Aula 15 Via Tiburtina, 205	Lunedì	14:00 – 17:30
Ricevimento:	Per appuntamento	in FUB, per email, per telefono
Sito web:	http://w3.uniroma1.it/IngClinFondinf	



Le funzioni

- ❑ Concetto di modulo di un programma
- ❑ Funzioni in C
- ❑ Introduzione di una funzione
- ❑ Esempio di funzione in C
- ❑ Prototipo di una funzione
- ❑ Definizione di funzione
- ❑ Chiamata di funzione
- ❑ Prototipo, definizione, chiamata
- ❑ Programma con funzioni
- ❑ Passaggio degli argomenti
- ❑ Variabili locali
- ❑ Esempi
- ❑ Libreria matematica

Concetto di modulo di un programma (1)



- ❑ Lo sviluppo di programmi **complessi** avviene tipicamente per composizione di **moduli**, ognuno dei quali esegue uno compito **semplice**
 - ❑ Con l'uso della **modularizzazione** (scomposizione di un programma in moduli) si ottengono normalmente programmi facilmente
 - **testabili/mantenibili** (sono possibili operazioni di test/modifica limitate a singoli moduli)
 - **leggibili/documentabili** (il programma si presenta come una composizione di compiti semplici)
- (... e si semplifica la **ripartizione** di un progetto tra più progettisti)

Concetto di modulo di un programma (2)



- ❑ La modularizzazione è particolarmente apprezzata quando almeno un modulo è **usato più volte** durante l'esecuzione del programma (tipicamente, in questo caso si ha anche una **compattazione** del codice sorgente)
- ❑ E' utile tentare di anticipare la modularizzazione alla fase di sviluppo dell'algoritmo per la soluzione di un problema



Funzioni in C (1)

- Un programma C si compone di **funzioni**
- E' disponibile una collezione *predefinita* di funzioni che possono essere usate *direttamente* in ogni programma (**libreria standard**)
- Per introdurre una **nuova funzione** occorre specificare:
 - una sequenza di **operazioni**
 - gli **argomenti** su cui operare
 - il **risultato** da restituire
- Un programma C viene modularizzato con l'uso di funzioni



Funzioni in C (2)

- ❑ Le istruzioni specificate in una funzione sono eseguite quando la funzione viene **attivata**
- ❑ La funzione **main** è **attivata** quando il programma viene messo in esecuzione
- ❑ Le altre funzioni sono attivate durante l'esecuzione del programma, tramite **chiamata (call)**
- ❑ Una funzione può **chiamare** una o più funzioni
- ❑ Una funzione può chiamare se stessa (**chiamata ricorsiva**)



C: Introduzione di una nuova funzione

- ❑ Nel programma, una funzione può essere chiamata solo dopo averla **dichiarata**
- ❑ La **dichiarazione** di una funzione avviene per mezzo di un **prototipo di funzione** nel quale vengono specificati **nome**, **argomenti** e **risultato** della funzione
- ❑ La **sequenza di istruzioni** eseguite da una funzione è invece specificata nella sua **definizione**
- ❑ La **chiamata** di una funzione avviene specificando il **nome** di questa e gli **argomenti** da usare durante l'esecuzione



Esempio di funzione C (1)

- Cubo di un intero: argomento `int`, risultato `int`
- Dichiarazione: **nome**, argomenti, risultato
`int cubo (int) ;`
- Chiamata: specificare **nome**, argomento
`a = cubo (b) ;` (`a` e `b` interi, produrrà `a = b^3`)
- Definizione: specificare **nome**, argomento su cui operare, risultato
`int cubo (int n)`
{
 `return n*n*n ;`
}

Esempio di funzione C (2)



```
...
int cubo( int ) ;          /* prototipo funzione cubo */
int main( )
{
    int num ;
    ...                    /* attivazione funzione cubo */
    printf( "cubo(%d) = %d", num , cubo( num ) ) ;
    ...
    return 0;
}                          /* definizione funzione cubo */
int cubo( int n )
{
    return n*n*n ;
}
```

C: Prototipo di funzione



- **Formato**

tipo-restituito **nome** (**lista-argomenti**) ;

- **tipo-restituito** è il tipo del valore restituito dalla funzione come risultato: se non viene restituito alcun tipo usare **void**
- **nome** è utilizzato nella chiamata e nella definizione: stesse regole date per il nome di una variabile
- **lista-argomenti** specifica un tipo ed un nome opzionale per ogni argomento della funzione; una virgola separa i vari argomenti; può essere vuota

C: Definizione di funzione



- **Formato (deve essere *coerente* con il prototipo)**
tipo-restituito nome (lista-argomenti)
{ corpo-funzione }
- **tipo-restituito** e **nome**: vedi prototipo
- **lista-argomenti** per ogni argomento della funzione specifica un **tipo** ed un **nome**; una virgola separa i vari argomenti; può essere vuota; *numero/ tipo di argomenti e loro ordine come nel prototipo*
- **corpo-funzione** specifica le azioni da eseguire: contiene dichiarazioni di variabili, istruzioni, etc. Per restituire un valore si usa (*almeno una volta*)
return espressione ;



C: Chiamata di funzione

- **Formato (deve essere *coerente* con il prototipo)**
nome (lista-espressioni)
- **nome**: vedi prototipo
- **lista-espressioni** contiene un'espressione per ogni argomento; espressioni separate da virgola; *numero/ tipo di espressioni e loro ordine come nel prototipo*
- **Chiamata di funzione inserita in un'istruzione.**
- Se la funzione restituisce un valore, la chiamata compare normalmente in un'espressione il cui valore è assegnato ad una variabile
- Esempi

funct1(alpha) ;

$y = 2 * \text{cubo}(x) + 7 ;$

C: prototipo, definizione, chiamata (1)



- `double power(double , unsigned int) ;`
- `double power(double base, unsigned int expo)`
`{`
`int h ;`
`double result = 1 ;`
`for(h = 1 ; h <= expo ; h++)`
`result = result * base ;`
`return result ;`
`}`
- `y = power(x , n) ;`

C: prototipo, definizione, chiamata (2)



- `void power(double , unsigned int) ;`
- `void power(double base, unsigned int expo)`
`{`
`int h ;`
`double result = 1 ;`
`for(h = 1 ; h <= expo ; h++)`
`result = result * base ;`
`printf("power(%lf,%u) = %lf" , base, expo, result) ;`
`return ; /* return può essere omesso */`
`}`
- `power(x , n) ;`

C: Programma con funzioni



Schema di riferimento *per questo corso*

- **Dichiarare le funzioni tramite i prototipi prima del main**
- **Definire le funzioni corrispondenti ai prototipi dopo il main**
- **Inserire le chiamate delle funzioni dove necessario**

C: Passaggio degli argomenti



- Gli argomenti presenti nel prototipo e nella definizione di una funzione vengono detti anche **argomenti (o parametri) formali**, distinguendoli così dagli argomenti effettivamente utilizzati durante la chiamata della funzione, detti **argomenti (o parametri) attuali**
- Nella chiamata **power(x , n)** gli argomenti attuali sono i valori delle variabili **x** ed **n** al momento della chiamata.
- Alla funzione vengono passati i *valori* di **x** ed **n**. La funzione non può modificare il valore di **x** ed **n**

C: Passaggio degli argomenti



- ❑ Le espressioni passate come parametri attuali vengono valutate, convertite ai tipi dichiarati nel prototipo e assegnate ai rispettivi parametri formali
- ❑ Le variabili coinvolte nell'espressione sono passate per *valore* e non possono essere modificate dalla funzione



C: Variabili locali

- ```
double power(double base, unsigned int expo)
{
 int h ;
 double result = 1 ;
 for(h = 1 ; h <= expo ; h++)
 result = result * base ;
 return result ;
}
```
- Variabili `h` e `result` sono dette **locali**: non sono visibili al di fuori del blocco della funzione `power`
- Nel blocco della funzione `power` non sono visibili le variabili dichiarate all'interno della funzione `main` o di altre funzioni

# Esempio: Funzione in un programma



```
int max(int , int) ; /* prototipo */

int main ()
{
 int num1, num2 ;
 printf("Inserire due interi\n") ;
 scanf("%d%d" , &num1 , &num2) ;
 printf("Max = %d\n " , max(num1 , num2)) ;
 return 0 ; /* chiamata */
}

int max (int n1 , int n2) /* definizione */
{
 if (n1 > n2) return n1 ;
 else return n2 ;
}
```



## C: Libreria matematica (1)

- Collezione di funzioni matematiche predefinite, utilizzabili inserendo la direttiva

`# include <math.h>`

- **Esempi**

`double sqrt( double x )`: radice quadrata non negativa di **x**

`double pow( double x , double y )`: **x** elevato alla potenza **y**

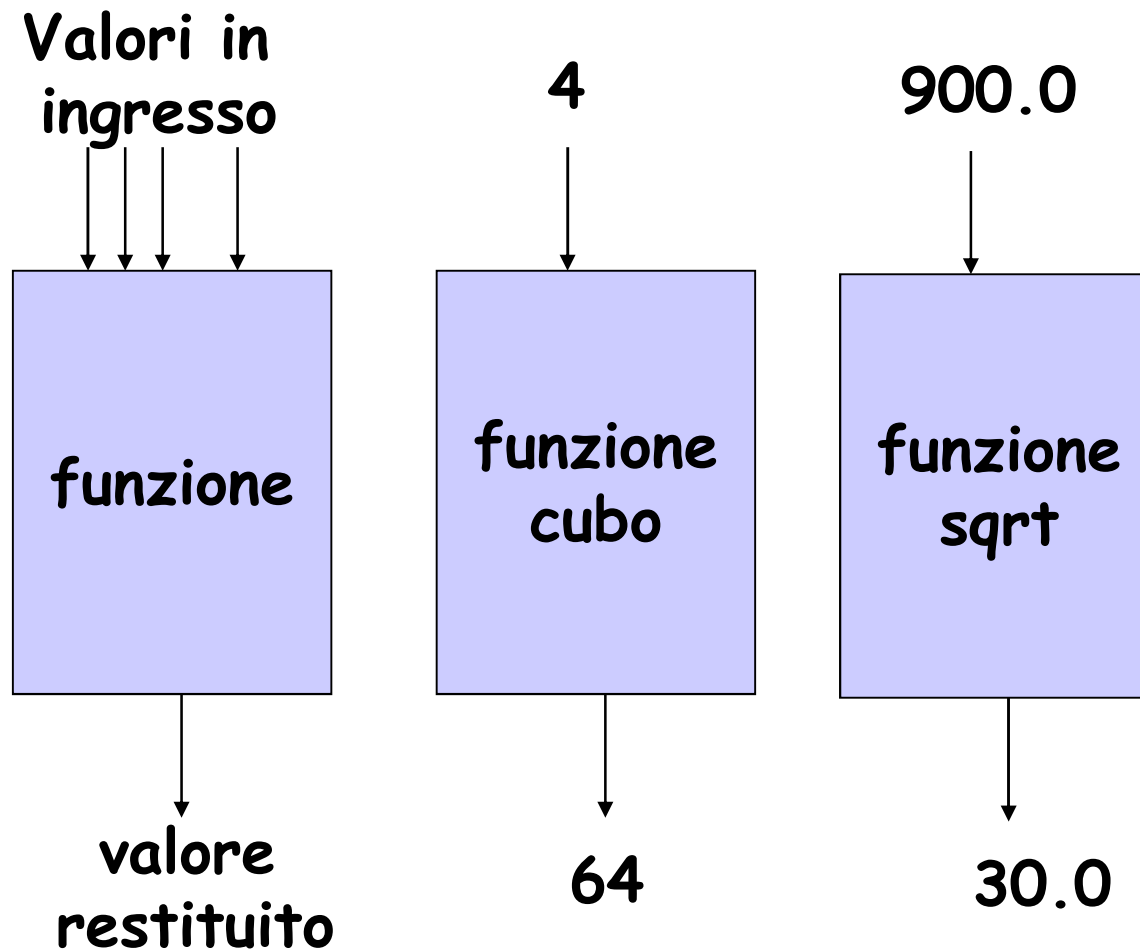
`double floor( double x )`: *intero* più grande non maggiore di **x**

`double ceil( double x )`: *intero* più piccolo non minore di **x**

# Funzioni come scatola nera



A partire da uno o più valori in ingresso le funzioni restituiscono un valore al modulo chiamante



# DevCpp: funzioni



E' possibile vedere le funzioni utilizzate nel programma. Per ogni funzione viene indicato nome, parametri e tipo del valore restituito

```
void merge(int *v1, int *v2, int *risultato, int m,
{
 int i=0,j=0,k=0;

 /* confronto e copio in modo ordinato finche' uno d.
 while(i<m && j<m)
 if (v1[i]<v2[j])
 risultato[k++]=v1[i++];
 else
```

**Andare alla scheda "Classi"**

# Esercizio 1



Scrivere un programma che, acquisiti da stdin 3 valori reali (float) **a**, **b**, **c**, usi due funzioni per determinare, rispettivamente, minimo e massimo tra i valori acquisiti, e visualizzi su standard output i risultati restituiti dalle funzioni





# Soluzione esercizio 1

```
/* definizione funzione min3 */
float min3(float x , float y , float z)
{
 float min ;

 if (x < y)
 min = x ;
 else
 min = y ;

 if (z < min)
 return z ;
 else
 return min ;
}
```



# Soluzione esercizio 1

```
/* definizione funzione max3 */
float max3(float x , float y , float z)
{
 float max ;

 if (x > y)
 max = x ;
 else
 max = y ;

 if (z > max)
 return z ;
 else
 return max ;
}
```

# Soluzione esercizio 1



```
#include <stdio.h>
#include <stdlib.h>

/* prototipo funzione min3 */
float min3(float , float , float) ;

/* prototipo funzione maxi3 */
float max3(float , float , float) ;

int main()
{
 float a , b , c ;

 printf("\nInserire a (float): ") ;
 scanf("%f" , &a) ;
 printf("\nInserire b (float): ") ;
 scanf("%f" , &b) ;
 printf("\nInserire c (float): ") ;
 scanf("%f" , &c) ;
 printf("\nminimo(%f , %f , %f) = %f\n" , a , b , c , min3(a , b , c)) ;
 printf("\nmassimo(%f , %f , %f) = %f\n" , a , b , c , max3(a , b , c)) ;
 system("pause") ;
 return 0 ;
}
```

# Esercizio 2



Progettare una funzione che, ricevuti un carattere (char)  $c$  e due valori reali (float)  $x$  e  $y$ , restituisca il reale (float) corrispondente all'operazione specificata da  $c$  ('+' sta per somma, '-' sta per differenza, '\*' sta per prodotto, '/' sta per divisione) eseguita tra  $x$  e  $y$ . In caso di operatore non valido (carattere  $c$  diverso dai quattro valori specificati sopra), la funzione restituirà il valore  $0.0$  e non effettuerà alcuna operazione tra  $x$  e  $y$ .

Inserire poi la funzione in un programma di prova, che provveda all'acquisizione da standard input degli argomenti della funzione, alla chiamata della funzione e alla visualizzazione su standard output del valore da questa restituito

## Soluzione esercizio 2



```
/* Definizione funzione operation */
```

```
float operation(char op , float opnd1 , float opnd2)
```

```
{
```

```
 if (op == '+')
```

```
 return (opnd1 + opnd2) ;
```

```
 else if (op == '-')
```

```
 return (opnd1 - opnd2) ;
```

```
 else if (op == '*')
```

```
 return (opnd1 * opnd2) ;
```

```
 else if (op == '/')
```

```
 return (opnd1 / opnd2) ;
```

```
 else
```

```
 return 0.0 ;
```

```
}
Università degli Studi "La Sapienza" – Fondamenti di Informatica
```

# Soluzione esercizio 2



```
#include <stdio.h>
#include <stdlib.h>

/* Prototipo funzione operation */
float operation(char , float , float);

int main()
{
 float x , y ;
 char op ;

 printf("Inserire carattere operatore ('+', '-', '*' o '/'): ");
 scanf("%c" , &op);

 printf("\nInserire 1o operando (float): ");
 scanf("%f" , &x);

 printf("\nInserire 2o operando (float): ");
 scanf("%f" , &y);

 printf("\n%f %c %f = %f\n\n" , x , op , y , operation(op , x , y));

 system("pause");
 return 0 ;
}
```

# Esercizio 3: somma di due frazioni



Scrivere un programma per **calcolare la somma di due frazioni**.

Il programma deve:

1. ricevere in input quattro interi (num1, den1, num2, den2) corrispondenti ai numeratori e ai denominatori delle frazioni da sommare
2. fornire in output numeratore e denominatore della frazione (eventualmente semplificata) risultante dalla somma dei due numeri.

# Esercizio 3: funzioni del programma



1. funzione **massimoComunDivisore** che calcola il massimo comun divisore di due interi
2. funzione **minimoComuneMultiplo** che calcola il minimo comune multiplo di due interi
3. funzione **main**



# Esercizio 3: massimoComunDivisore



Calcolare il massimo comune divisore tra due numeri  $a$  e  $b$  in ingresso.

- **Algoritmo di Euclide:** permette di determinare il M.C.D. di due numeri naturali in modo molto più efficiente che non elencando tutti i divisori dei due numeri considerati.

```
do {
 r = a % b;
 a = b;
 b = r;
} while (r != 0)
```

Il massimo comun divisore è  $a$

Esempio      $a = 12$   $b = 15$

|    |        |         |        |
|----|--------|---------|--------|
| 1) | $r=12$ | $a= 15$ | $b=12$ |
| 2) | $r=3$  | $a=12$  | $b=3$  |
| 3) | $r=0$  | $a=3$   | $b=0$  |

**MCD = 3**

# Esercizio 3: minimoComuneMultiplo



- La funzione `minimoComuneMultiplo` utilizza la funzione `massimoComunDivisore`
- Il minimo comune multiplo è uguale al prodotto dei due numeri diviso il loro massimo comun divisore,  
 $mcm(a,b) = a*b / MCD(a,b)$

# Esercizio 3: main



La funzione **main**:

- legge l'input
- calcola il risultato usando gli altri moduli
- verifica se il risultato deve essere semplificato
- stampa il risultato

Per effettuare la somma di due frazioni si dovrà

1. calcolare il mcm dei due denominatori che costituisce il denominatore della frazione\_somma
2. calcolare i numeratori delle frazioni equivalenti a quelle date con il nuovo denominatore
3. sommare i nuovi numeratori per avere il numeratore della frazione\_somma

Per semplificare una frazione si dovrà calcolare il MCD tra numeratore e denominatore



## Soluzione esercizio 3

```
int minimoComuneMultiplo (int n1, int n2)
{
 return n1 * n2 / massimoComunDivisore(n1, n2);
}
```

```
int massimoComunDivisore (int n1, int n2)
{
 int resto;

 do {
 resto = n1 % n2;
 n1 = n2;
 n2 = resto;
 } while (resto != 0);

 return n1;
}
```



## Soluzione esercizio 3

```
void stampaRisultato (int n1, int d1, int n2, int d2, int ns, int ds)
{
 printf("%d/%d + %d/%d = %d/%d\n", n1, d1, n2, d2, ns, ds);
}
```

## Soluzione esercizio 3



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

```
#include <stdlib.h>
```

```
int massimoComunDivisore (int n1, int n2);
```

```
int minimoComuneMultiplo (int n1, int n2);
```

```
void stampaRisultato (int n1, int d1, int n2, int d2, int ns, int ds);
```

```
/* scrive il risultato della somma */
```

# Soluzione esercizio 3



```
int main ()
{
 int num1, den1, num2, den2 ;
 int num_sum, den_sum, semplifica;

 printf("Inserire quattro interi [num1 den1 num2 den2]\n");
 scanf("%d%d%d%d", &num1, &den1, &num2, &den2);

 den_sum = minimoComuneMultiplo(den1, den2);
 num_sum = num1*(den_sum/den1) + num2*(den_sum/den2);

 semplifica = massimoComunDivisore(num_sum, den_sum);
 if (semplifica != 1) {
 num_sum /= semplifica;
 den_sum /= semplifica;
 }
 stampaRisultato(num1, den1, num2, den2, num_sum, den_sum);

 system("pause");

 return 0;
}
```

$$\frac{3}{2} + \frac{5}{4} = \frac{3 * (4 / 2) + \dots}{4}$$

*/\* denominatore comune \*/*