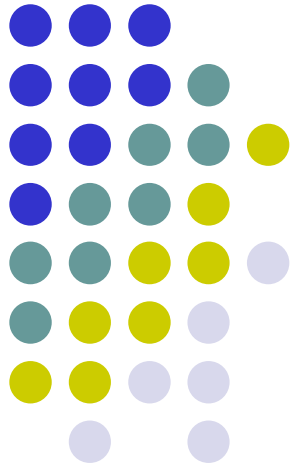
# Laboratorio di Calcolatori 1

## Corso di Laurea in Fisica

## A.A. 2006/2007



Dott.Davide Di Ruscio

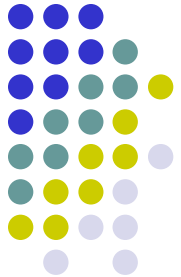
Dipartimento di Informatica

Università degli Studi di L'Aquila

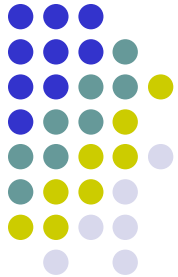
Lezione del 15/05/08

# Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele



# Sommario (II parte)



Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

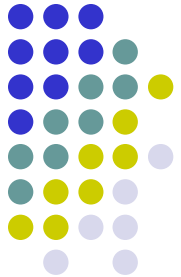
RIFERIMENTI

Ceri, Mandrioli, Sbattella

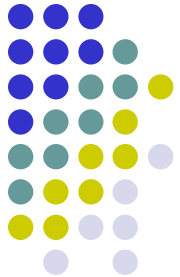
Informatica arte e mestiere

McGraw-Hill

# I sottoprogrammi in C

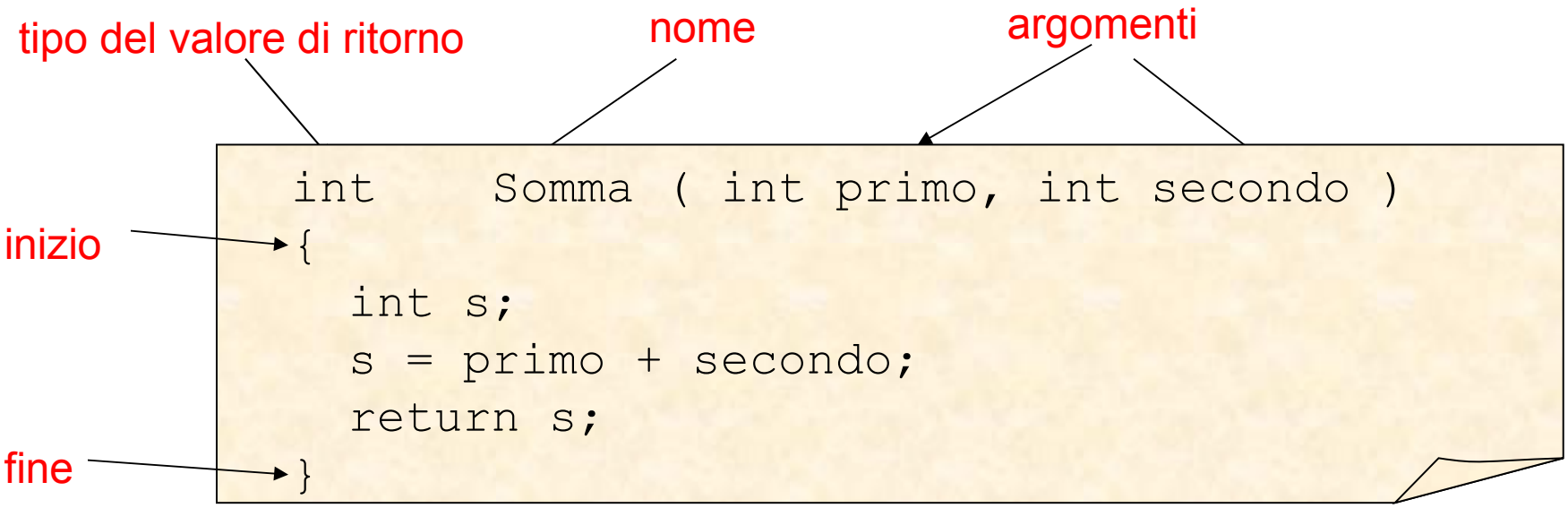


- Il concetto di sottoprogramma
- Struttura completa di un programma C
- Le funzioni
  - Esecuzione delle funzioni e passaggio dei parametri
- Le procedure
- Il passaggio dei parametri per indirizzo
- Aspetti avanzati nell'uso dei sottoprogrammi

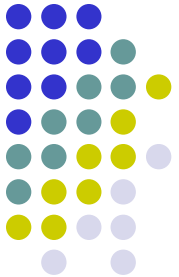


# Unità fondamentale: *funzione*

Mediante le funzioni si possono definire operazioni complesse a partire da istruzioni elementari



# Le procedure



- Le procedure sono semplicemente delle funzioni che non restituiscono alcun valore
- Esempio: ordina un vettore, stampa n trattini, …
- Il loro tipo di ritorno è `void`
- Possono agire:
  - modificando variabili globali
  - modificando parametri passati per indirizzo (vedremo fra poco)
  - intervenendo in fase di input/output
- Vengono chiamate come fossero un'istruzione del C.

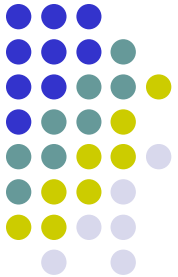
Esempio:

```
int x=0;

void p (int a);
main {
   int a;
   scanf ("%d",&a);
   p(a);
}
```

```
void p (int a){
   x=x+a;
}
```

La procedura p modifica il valore della variabile globale x

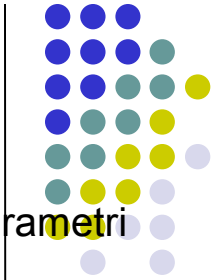
# Altro esempio



```
#include<stdio.h>
main()
{
   void stampatratt(int n);
   int k;

   scanf("%d",&k);
   stampatratt(k);
}

void stampatratt(int n)
{
   int i;

   for (i=1; i<=n; i=i+1)
    printf("%c",'-');
}
```

# Passaggio di parametri

```
/* Domanda: cosa stampa il seguente programma se vengono letti in
   input i numeri 1 e 3*/
#include <stdio.h>
main()
{
   void scambia(int x, int y);
   int a,b;

   scanf("%d%d",&a,&b);
   scambia(a,b);
   printf("%d %d",a,b);
}
void scambia(int x, int y)
{
   int aux;
   aux=x;
   x=y;
   y=aux;
}
Risposta: 1 3 !
```

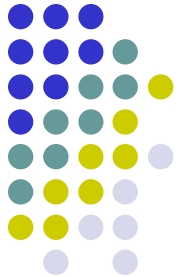


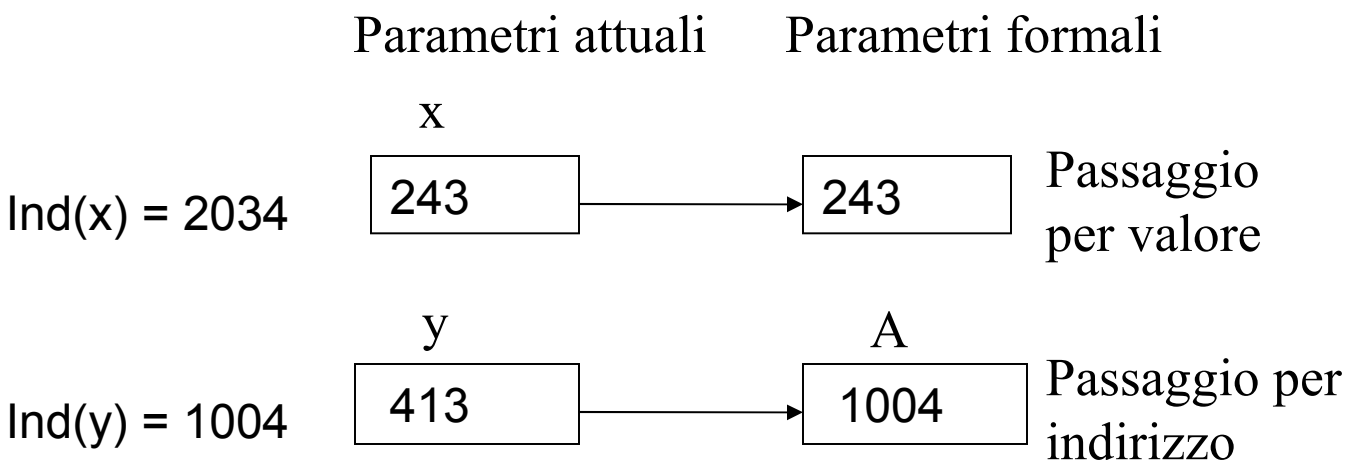
- All'atto della chiamata i parametri attuali vengono valutati e **copiati** nell'ordine ai parametri formali
- Questo tipo di passaggio è detto per **valore**, ed è tale che la funzione non modifica il valore della variabile passata.

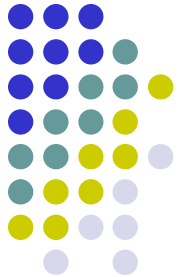
Esempio:

```
int f(int a, int b);

main(){
   int x, somma;
   scanf ("%d",&x);
   somma = f(x, 3); /*la funzione non modifica il valore di x*/
   printf ("%d %d",x,somma);
}

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

# Passaggio parametri per indirizzo

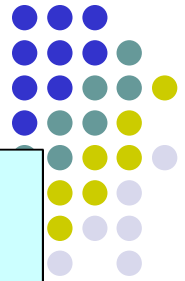




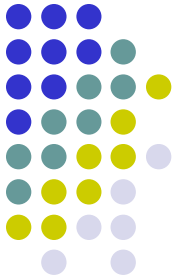


***In C la modalità di passaggio dei parametri a un sottoprogramma è sempre quella di passaggio per valore. Per simulare il passsaggio per indirizzo:***

- utilizzare il costruttore *puntatore* per la definizione dei parametri formali della funzione;
- usare l'operatore di dereferenziazione di puntatore (operatore * o –>) all'interno del corpo della funzione;
- passare al momento della chiamata della funzione come parametro attuale un indirizzo di variabile (usando l'operatore &).
- in altre parole si passano gli indirizzi delle variabili da modificare

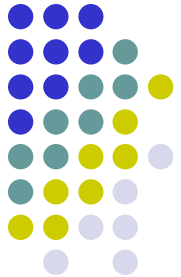


```
/*Ed ora cosa stampa il seguente programma se vengono letti in
   input i numeri 1 e 3?*/
#include <stdio.h>
main()
{
   void scambia(int *x, int *y);
   int a,b;
   scanf("%d%d",&a,&b);
   scambia(&a,&b);
   printf("%d %d",a,b);
}
void scambia(int *x, int *y)
{
   int aux;

   aux=*x;
   *x=*y;
   *y=aux;
}
Risposta: 3 1 !!!
```



```
//Altro esempio:

   int f(int* a);

   main(){
      int x, doppio;
      scanf (“%d”,&x);
      doppio = f(&x); /*la funzione ora modifica il valore di x*/
      printf (“%d %d”,x,somma);
   }

   int f (int *a){
      *a=*a+*a;
      return *a;
   }
```

# Visibilità delle dichiarazioni

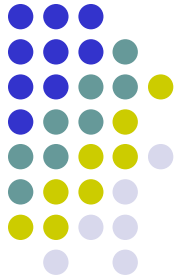


- Le dichiarazioni locali di una funzione (o del main) sono visibili soltanto all'interno della funzione stessa
- Le dichiarazioni globali sono visibili al main e a tutte le altre procedure e funzioni
- In caso di omonimia le dichiarazioni locali hanno priorità e ricoprono quelle globali

```c
#include <stdio.h>

main()
{
   int fatt(int x);
   int n,k;

   scanf("%d%d",&n,&k);
   bin= fatt(n)/(fatt(k)*fatt(n-k));
   printf("%d",bin);
}

int fatt(int x)
{
   int f,i;

   f=1;
   for (i=2; i<=x; i=i+1)
        f=f*i;
   return(f);
}
```

```c
#include <stdio.h>

int fatt(int x);

main()
{
   int n,k;

   scanf("%d%d",&n,&k);
   bin= fatt(n)/(fatt(k)*fatt(n-k));
   printf("%d",bin);
}

int fatt(int x)
{
   int f,i;

   f=1;
   for (i=2; i<=x; i=i+1)
    f=f*i;
   return(f);
}
```

# Visibilità delle variabili

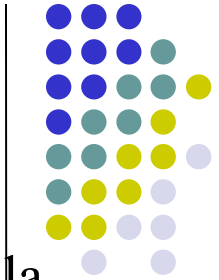


- Da quanto sottolineato le variabili locali hanno precedenza come visibilità rispetto alle variabili globali.
  - In una funzione non è visibile una variabile globale avente lo stesso identificatore di una variabile locale.
- I parametri formali delle funzioni sono variabili locali alla funzione.
- Esempio:

```
int x=0;

int g(int a) {
   int x=4;
   return a+x;
}
```

La funzione g non ha visibilità della variabile globale x

# Parametri di tipo array



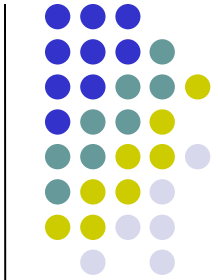
- L'indirizzo di base dell'array viene passato "per valore" alla funzione.

**typedef double** TipoArray[MaxNumElem]

- Le tre testate di funzione riportate di seguito sono equivalenti:

```
/* n è la dimensione dell'array passato */
double   sum(TipoArray a, int n)
double   sum(double *a,  int n)
double   sum(double  a[ ], int n)
```

NOTA: In C il passaggio dei parametri è **sempre e solo per valore,** tranne per i **vettori** che sono passati **sempre e solo per indirizzo**



```
double  mul(double  a[], int    n)
/* n è la dimensione dell'array
   passato */
{
   int  i;
   double   ris;

   ris = 1.0;
   for (i=0; i < n; i=i+1)
    ris = ris * a[i];
   return ris;
}
```

- v array di 50 elementi:

  *Chiamata  Valore calcolato e restituito*

  mul(v, 50)  v[0]*v[1]* ... *v[49]

  mul(v, 30)  v[0]*v[1]* ... *v[29]

  mul(&v[5], 7) v[5]*v[6]* ... *v[11]

  mul(v+5, 7)  v[5]*v[6]* ... *v[11]

# Esercizi

Scrivere un programma che utilizzi una funzione per

3. convertire i caratteri minuscoli di una parola in maiuscolo
   (ricorda che 65->'A', 97->'a')

```
void converti(char *a)
```

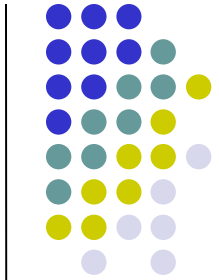
7. restituire la lunghezza di una stringa

```
int strLength(char *a)
```

- confrontare due stringhe; la funzione ritorna 0 se sono uguali, 1 se la prima è maggiore della seconda e -1 se la seconda è maggiore della prima

```
int isEqual(char *a, char *b)
```

# Uso interscambiabile di procedure e funzioni



```
intf(int par1)
{
    ...
    return risultato;
}
```

- Essa può essere trasformata nella procedura seguente:

```
void f(int par1, int  *par2)
{
    ...
    *par2 = risultato;
}
```

- Successivamente, una chiamata come:

  y = f(x);

  verrà trasformata in:

  f(x, &y);