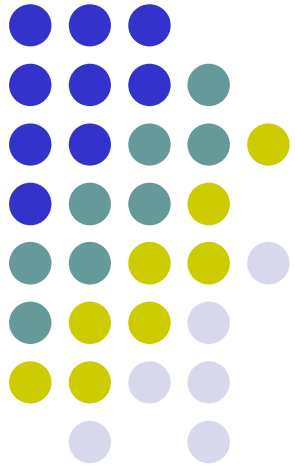
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica

A.A. 2006/2007



Dott.Davide Di Ruscio

Dipartimento di Informatica

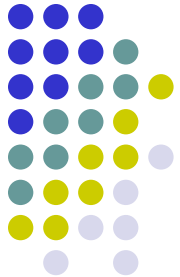
Università degli Studi di L'Aquila

Lezione del 20/02/07

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

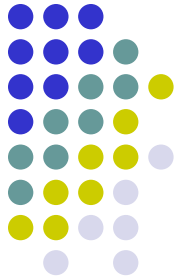
RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

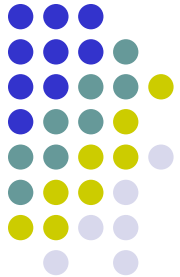
McGraw-Hill

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

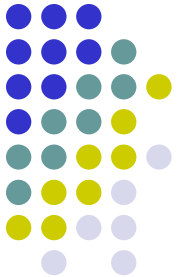
Il concetto di sottoprogramma



- Consideriamo un programma che dati due numeri interi *n* e *k* forniti in input calcola il loro coefficiente binomiale, ossia la quantità

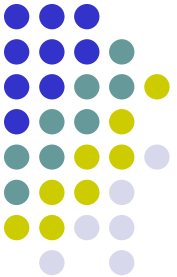
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Secondo quanto visto finora, ecco un possibile esempio

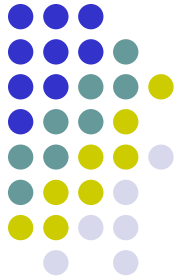


```
#include <stdio.h>

main()
{
   int n,k,f1,f2,f3,i,bin;

   scanf("%d%d",&n,&k);
   f1=1;
   for (i=2; i<=n; i=i+1)
        f1=f1*i;
   f2=1;
   for (i=2; i<=k; i=i+1)
        f2=f2*i;
   f3=1;
   for (i=2; i<=n-k; i=i+1)
        f3=f3*i;
   bin=f1/(f2*f3);
   printf("%d",bin);
}
```

Avendo a disposizione un'ipotetica funzione *fatt* che dato un generico intero *x* in input restituisce il *fattoriale* di x, non sarebbe meglio una soluzione simile alla seguente?

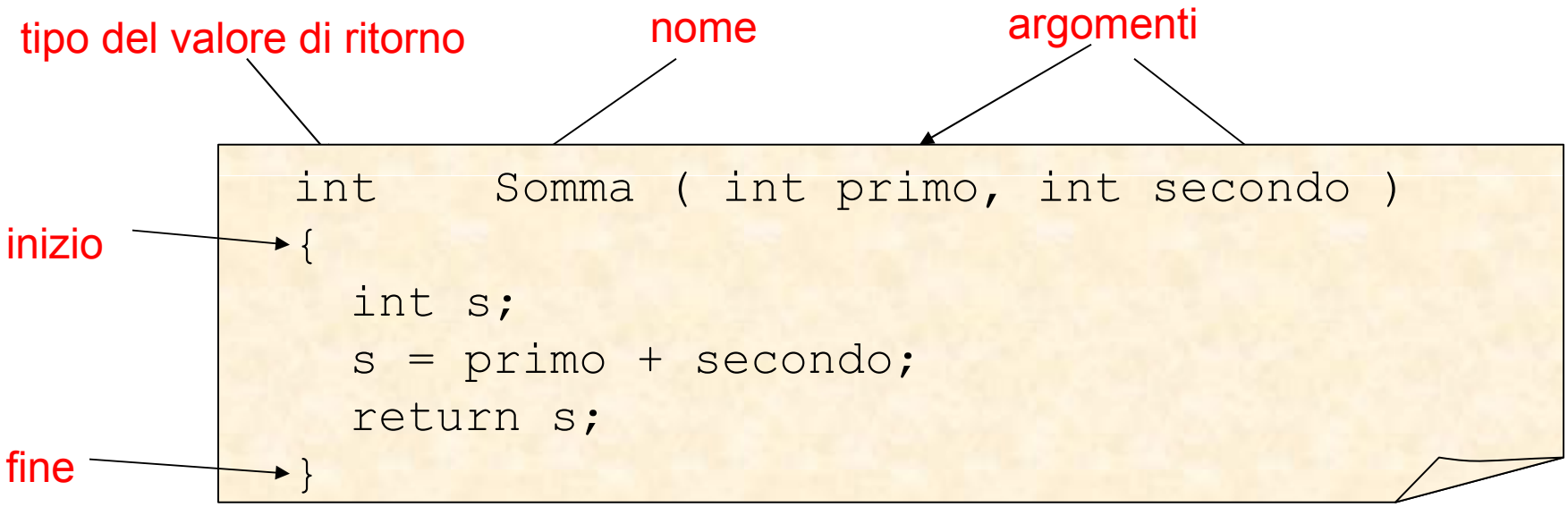


```
#include <stdio.h>

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

   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);
}
```

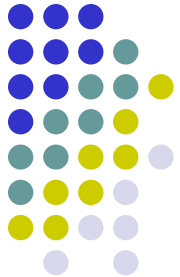


Unità fondamentale: *funzione*

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



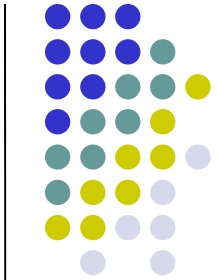
Una funzione speciale: main



main: è la funzione da cui inizia l'esecuzione del programma.

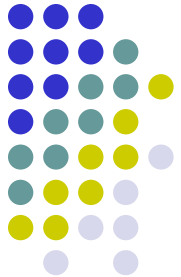
```
main()
{
      .....
}
```

Il concetto di sottoprogramma



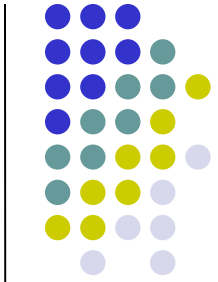
- I sottoprogrammi sono lo strumento per realizzare astrazioni sulle operazioni
- Due tipi (concettualmente):
  - Funzioni
  - Procedure

Struttura di un programma C



- direttive;
- **parte dichiarativa globale**
  - contiene la dichiarazione di tutti gli elementi che sono *condivisi* dal programma principale e dai sottoprogrammi.
- `main` (completo)
- definizioni di sottoprogrammi – funzioni o procedure –, a seguire il programma principale. Il main è un -particolare- sottoprogramma tra gli altri
- il `main` e ciascun sottoprogramma possono usare tutti e soli gli elementi che sono stati dichiarati nella loro propria parte dichiarativa, nonché quelli che sono stati dichiarati nella parte dichiarativa globale

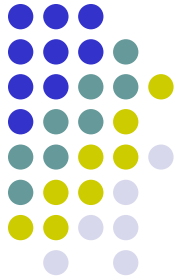
Funzioni



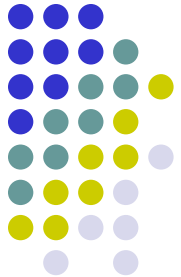
- *Definizione* (semplificata) di una funzione:
  - **testata** (*header*);
  - due parti, racchiuse fra parentesi graffe:
    - la parte dichiarativa (detta **parte dichiarativa locale**);
    - la parte esecutiva (detta **corpo** della funzione).

- La testata contiene:
  - tipo del risultato,
  - identificatore del sottoprogramma,
  - **lista dei parametri -formali-** cui la funzione viene applicata con il relativo tipo;
  - (dominio e codominio della funzione)
  - Ogni parametro formale è un identificatore di tipo seguito da un identificatore.
  - Una funzione non può restituire array o funzioni ma può restituire un puntatore a qualsiasi tipo.

Esempio di funzione



```
int  RadiceIntera (int par)
{
     int cont;

     cont = 0;
     while (cont*cont <= par)
             cont = cont + 1;
     return (cont - 1);
}
```



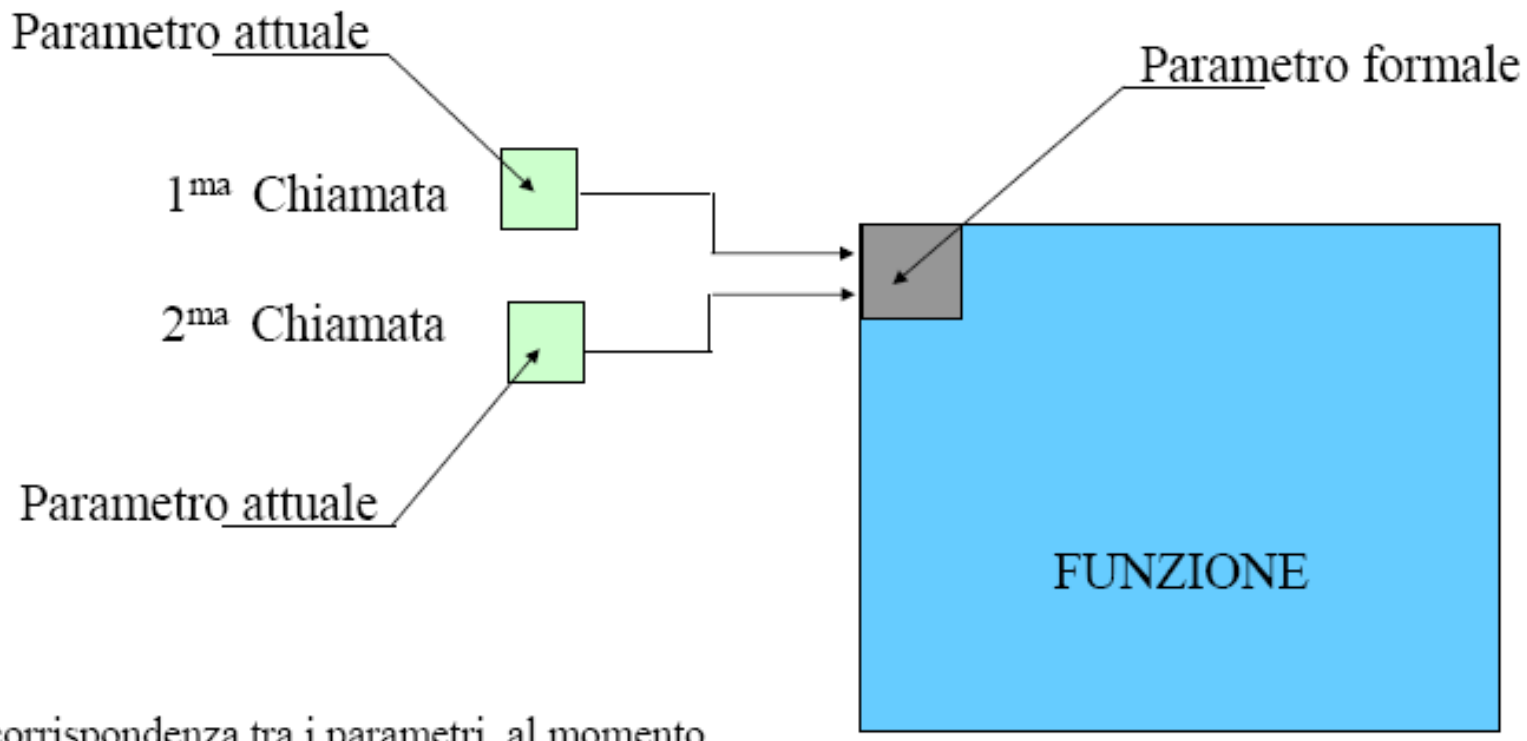
- Le funzioni ed il main possono anche essere definite su files differenti e quindi compilati separatamente.

- Le funzioni devono essere scritte in generale per la soluzione di una classe di problemi, per questo motivo devono poter lavorare su dati che, per ogni attivazione, possono assumere valori diversi.

- Il meccanismo del passaggio di parametri risolve tale problema, ed in generale permette lo scambio di informazioni tra le funzioni.

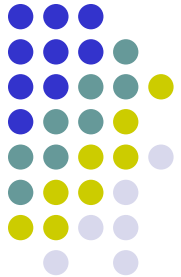
- Gli argomenti che vengono specificati nelle funzioni possono essere considerati come oggetti, per cosı dire, vuoti (parametri formali) che vengono riempiti all'atto di ciascuna attivazione, dei valori effettivi su cui la funzione deve lavorare (parametri attuali).

Passaggio di parametri



La corrispondenza tra i parametri, al momento della chiamata, si ottiene associando agli elementi della lista dei parametri formali i corrispondenti parametri attuali.

L'istruzione return



- La funzione restituisce un solo risultato del tipo dichiarato nell'header della funzione tramite l'istruzione

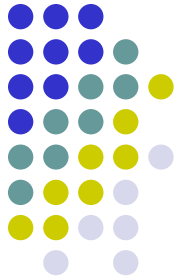
  `return` espressione`;`

- L'istruzione return provoca anche l'uscita dalla funzione

- La **chiamata ad una funzione** è in effetti una espressione (o una sottoespressione di un'espressione più complessa) avente il tipo dichiarato nell'header della funzione.

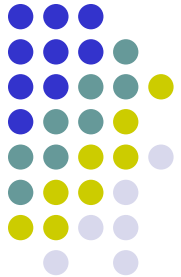
- Esempio: `x = f(x,1) * f(3,y)`

Chiamata delle funzioni

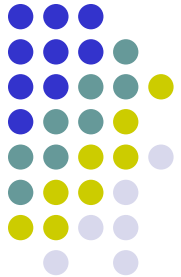


- I parametri attuali possono anche essere delle espressioni
- All'atto della chiamata i parametri attuali vengono valutati ed i loro valori assegnati nell'ordine ai parametri formali
- La funzione come effetto della sua esecuzione richiesta dalla chiamata restituisce il valore dell'espressione corrispondente all'istruzione return
- Tale valore viene sostituito direttamente nell'espressione al posto della chiamata alla funzione

Esempi di chiamate



```
x = sin(y) – cos(PiGreco – alfa);
x = cos(atan(y) – beta);
x = sin(alfa);
y = cos(alfa) – sin(beta);
z = sin(PiGreco) + sin(gamma);

RisultatoGestione = FatturatoTotale(ArchivioFatture)-SommaCosti(ArchivioCosti);

Det1 = Determinante(Matrice1);
Det2 = Determinante(MatriceInversa(Matrice2));

TotaleAssoluto = Sommatoria(Lista1) + Sommatoria(Lista2);
ElencoOrdinato = Ordinamento(Elenco);
OrdinatiAlfabeticamente = Precede(nome1, nome2);
```

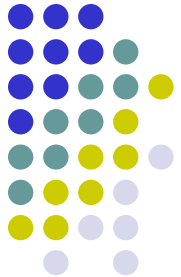


```
x = sin(atan(y) – acos(z));
```

Equivale a :

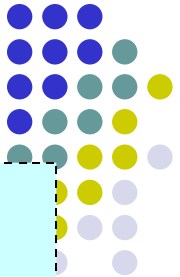
```
temp1 = atan(y); temp2 = acos(z);
temp3 = temp1 – temp2
x = sin(temp3);
```

Prototipo delle funzioni



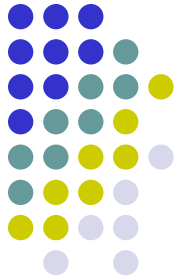
- All'interno di un programma C una funzione può essere chiamata purché risulti definita oppure dichiarata.
- *definizione* e *dichiarazione* non sono termini equivalenti
- La dichiarazione di una funzione (**prototipo**) si limita a richiamarne la testata.
- Utile quando la chiamata di una funzione precede, nel codice, la definizione della funzione, oppure le funzioni utilizzate da un programma sono definite in file propri del sistema C (e.g., le funzioni di libreria).
- Aiuta il compilatore ed è buono stile di programmazione

Esempio con definizione



```
#include <stdio.h>

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

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

main()
{
   int n,k,bin;

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

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.

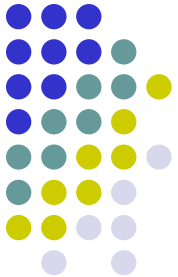
Esempio:

```
int x=0;

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

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

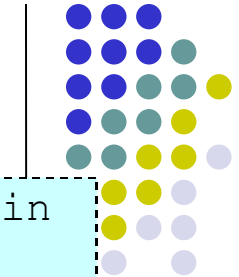
La procedura p modifica il valore della variabile globale x

Altro esempio



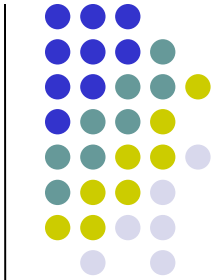
```
#include<stdio.h>
main()
{
   void stampatratt(int n);
   int k;

   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 !

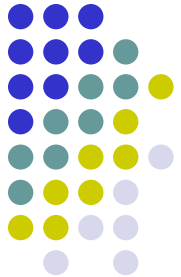


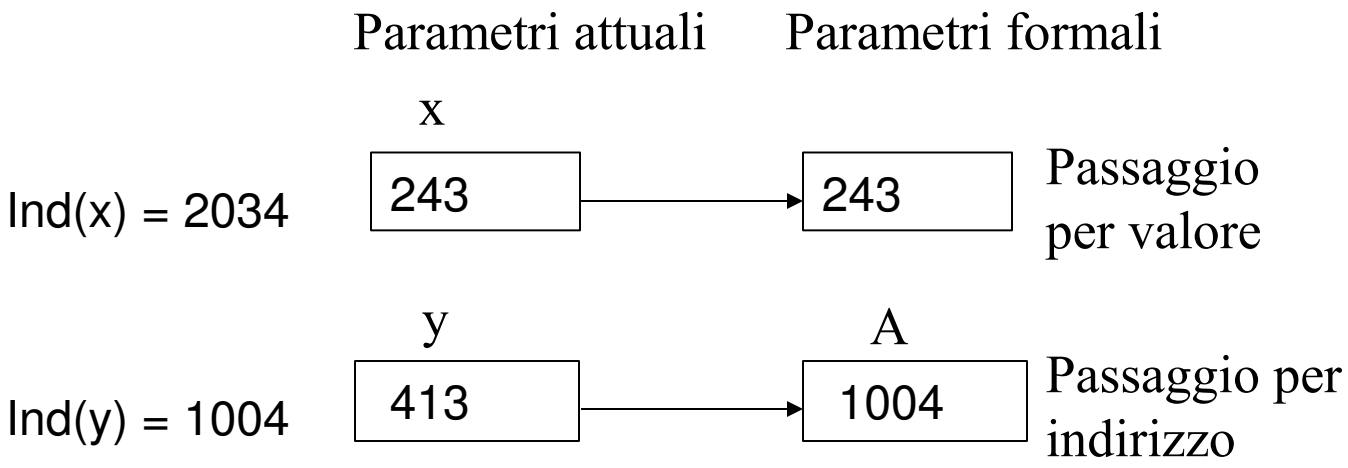
- All'atto della chiamata i parametri attuali vengono valutati e **copiati** nell'ordine ai parametri formali
- Questo tipo di passaggio è detto per **valore**, ed è tale che la funzione non modifica il valore della variabile passata.

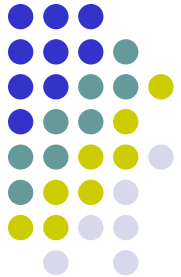
Esempio:

```
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

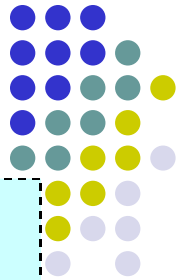




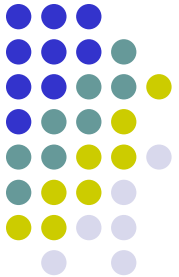


In C la modalità di passaggio dei parametri a un sottoprogramma è sempre quella di passaggio per valore. Per simulare il passsaggio per indirizzo:

- 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;
   }
```