Fondamenti di Informatica
Ingegneria Clinica
Lezione 3/12/2009

Raffaele Nicolussi
FUB - Fondazione Ugo Bordoni
Via B. Castiglione 59 - 00142 Roma

Legame per valore

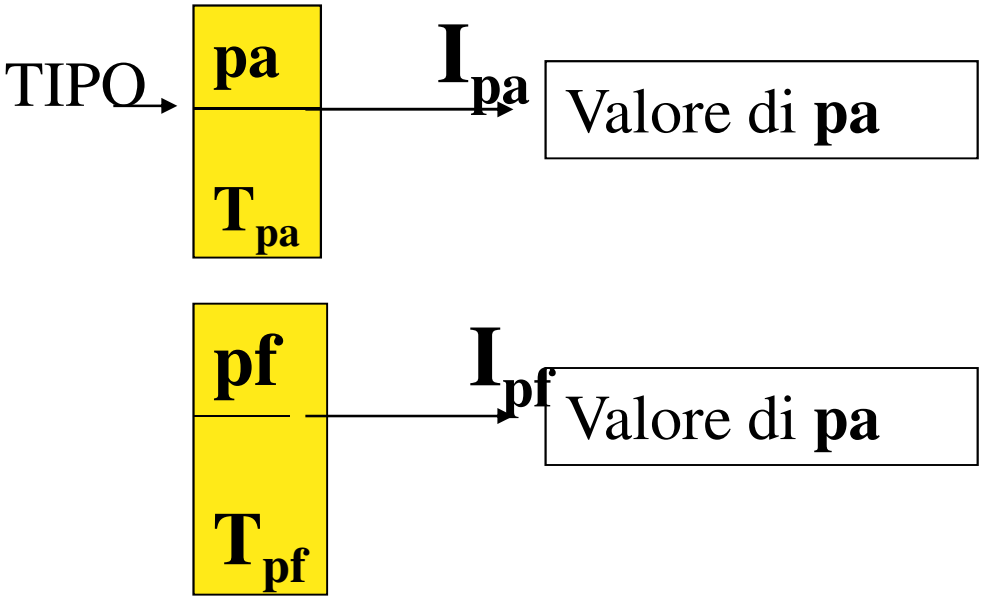
- I valori dei parametri attuali non sono modificati
- Il parametro formale (pf) è locale a tutti gli effetti, ossia le modifiche che avvengono su di esso non si ripercuotono a livello dell'unità chiamante
  - pa viene valutato prima della chiamata
  - il valore di pa viene assegnato a pf
  - le operazioni della chiamata non producono nessun effetto su pa

Legame per valore

- **Legare per valore pa a pf significa che**
  - **Dopo la valutazione di pa, viene creata una locazione di indirizzo pf**
  - **In tale locazione viene memorizzato il valore ottenuto valutando pa**
- **In pratica**
  - **Viene fatta una copia di pa (che è pf) e su quello si lavora**
  - **Alla fine dell'esecuzione della funzione, la memoria riservata a pf VIENE RILASCIATA (=liberata e resa disponibile) e il suo valore si perde (non viene visto fuori della funzione chiamata)**

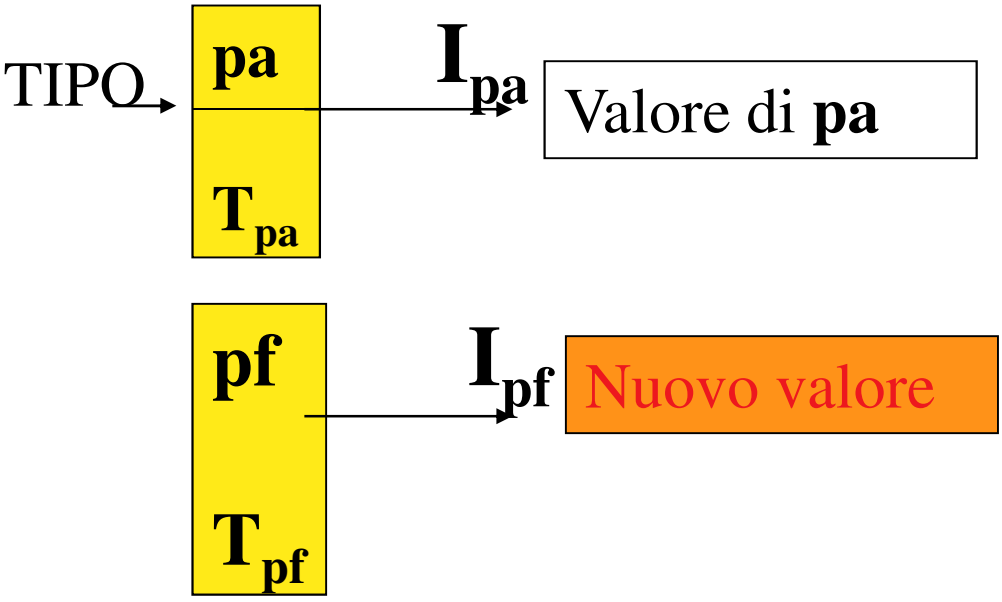
Legame per valore

Momento dell'attivazione di P



È creata una locazione per pf

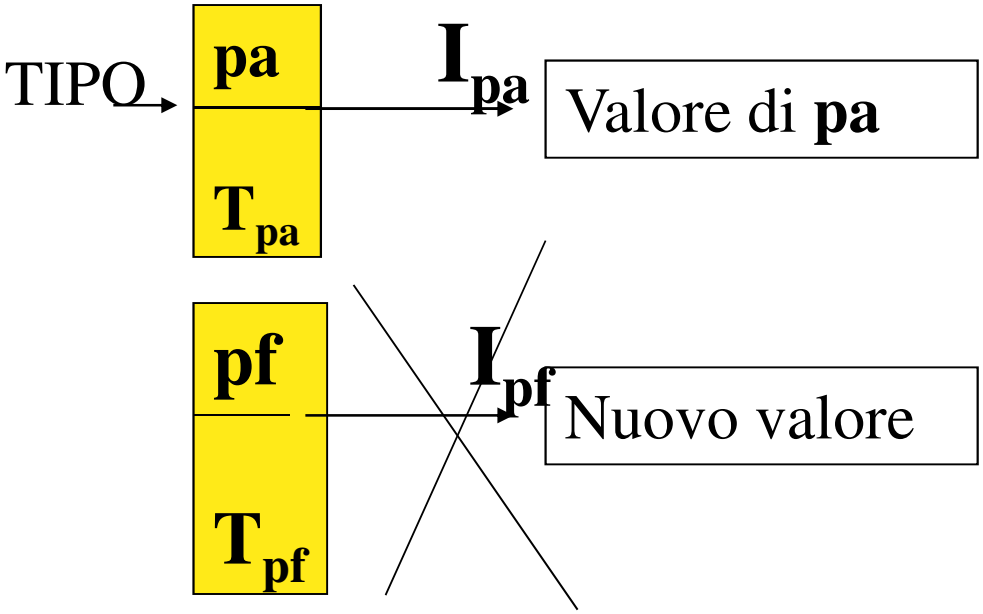
Durante l'esecuzione di P



I_{pa} , I_{pf} indirizzi

Legame per valore (2)

Fine dell'attivazione di P



Legame per riferimento (o variabile o indirizzo)

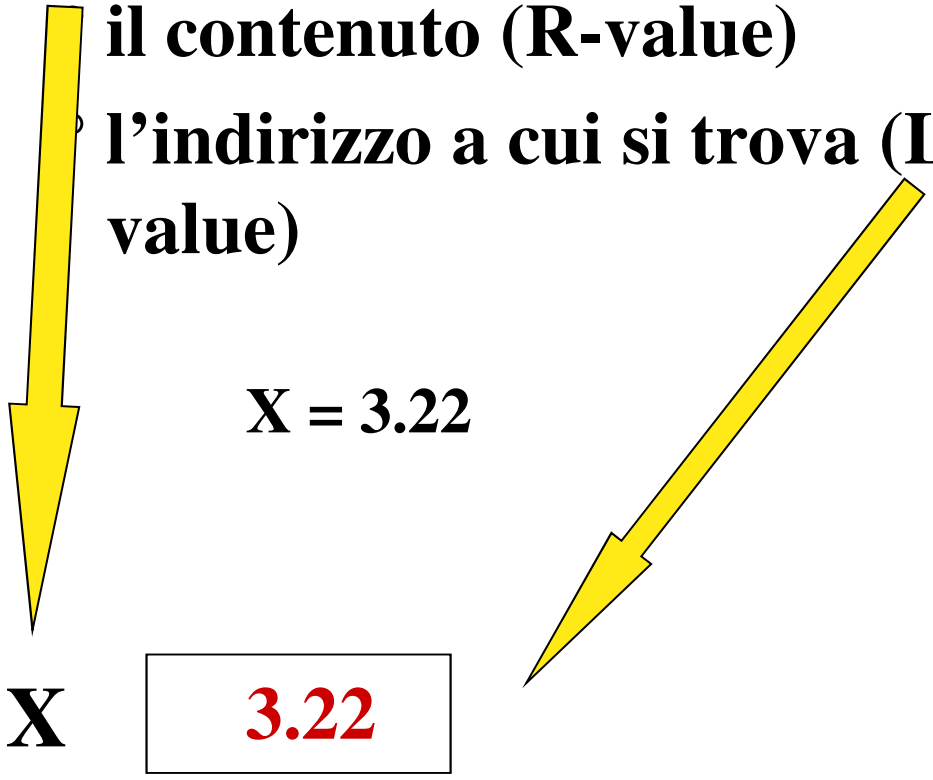
- La chiamata della funzione comporta la (eventuale) modifica del valore dei parametri attuali.
- Se pa è il parametro attuale della funzione chiamante P
  - il parametro formale pf si riferisce direttamente alla locazione di memoria identificata da pa
  - ogni cambiamento di pf si ripercuote così su pa

- Le azioni della chiamata sono fatte direttamente su pa e la chiamante vedrà le modifiche fatte su pa

- Al termine dell'attivazione il legame tra pf e pa viene perso

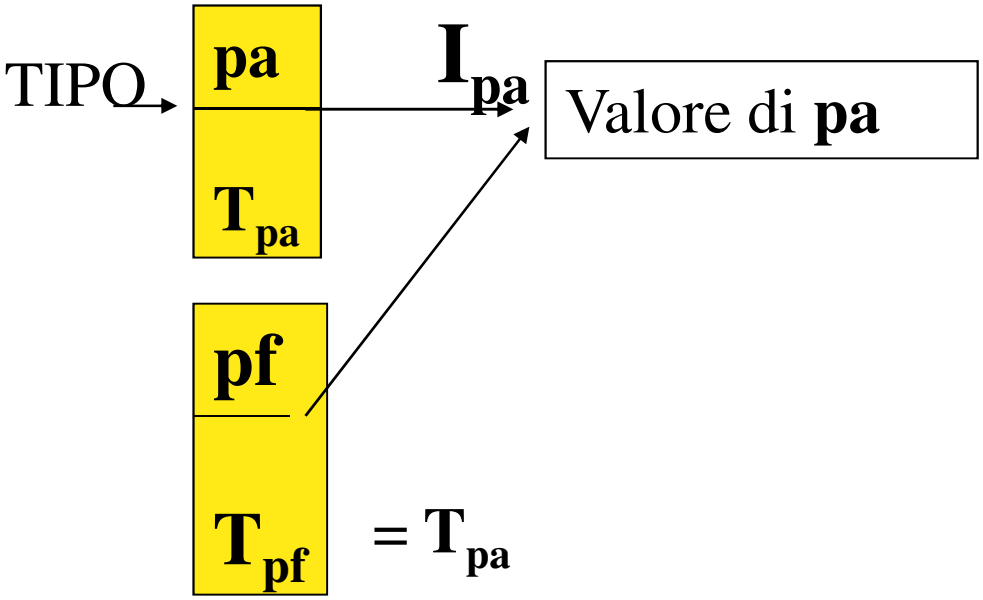
Una *variabile non è solo un sinonimo per un dato* come in matematica

- **è un'astrazione della cella di memoria associata a due diverse informazioni:**
  - **il contenuto (R-value)**
  - **l'indirizzo a cui si trova (L-value)**

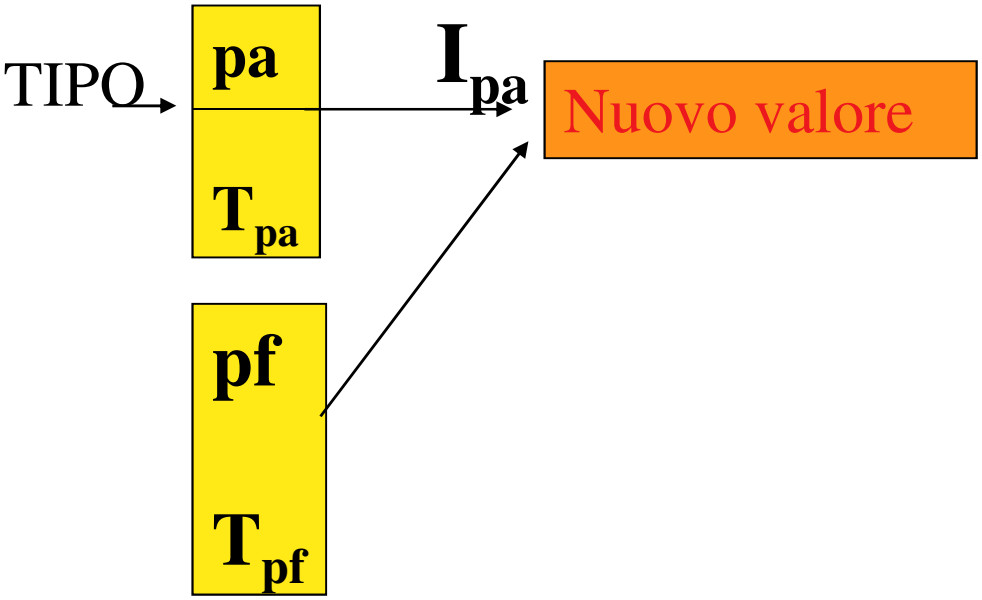


Legame per riferimento (variabile)

Momento dell'attivazione di P

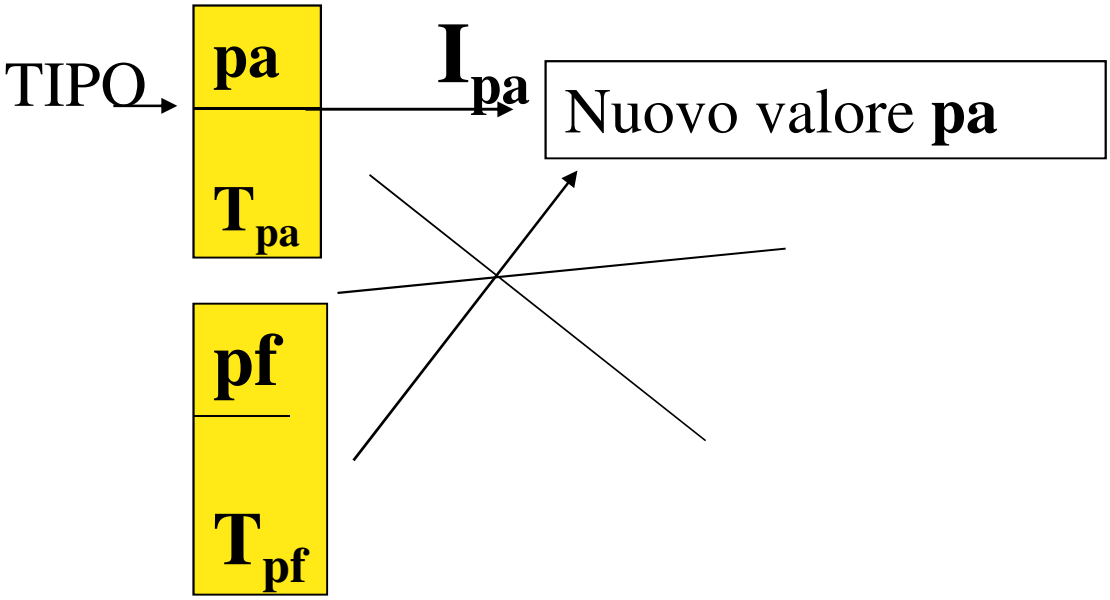


Durante l'esecuzione di P



Legame per riferimento (variabile)

Fine dell'attivazione di P



- **In C i legami tra parametri formali e attuali sono per valore**
  - **i valori dei parametri attuali nella funzione chiamante non vengono alterati dalla esecuzione della funzione chiamata.**
- **Esiste un modo per effettuare la chiamata per riferimento**
  - **Non è diretta**

- **Il passaggio di parametri per indirizzo sarà spiegato prossimamente su queste slide … ☺**

Blocchi

- Un programma C può essere decomposto in **blocchi**
  - **Un blocco è un'istruzione composta racchiusa tra parentesi graffe che può includere delle dichiarazioni**

```
{
   <dichiarazioni>
   <istruzioni>
   }
```

  - **Le variabili possono essere dichiarate all'interno di blocchi diversi con nomi diversi o con lo stesso nome.**

Blocchi

- I blocchi possono essere sia paralleli che nidificati

```
{
  {   }
}
{
}
```

- **Una funzione non può essere mai dichiarata all'interno di un altro blocco funzione**

Caratteristica delle variabili

- **Campo d'azione (scope)**
- **Tempo (o ciclo) di vita**
- **Tipo**
- **Valore**

- **Tipo: specifica la classe di valori che la variabile può assumere (e quindi gli operatori applicabili)**

- **Valore: è rappresentato (secondo la codifica adottata) nell'area di memoria associata alla variabile**

Caratteristica delle variabili

- **Tempo/Ciclo di vita:**
  - **è l'intervallo di tempo in cui rimane valida l'associazione simbolo/indirizzo fisico**

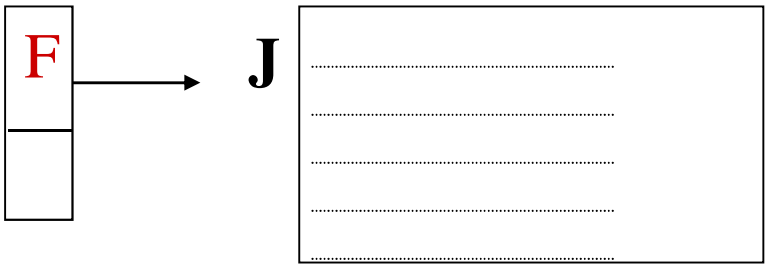
- **Campo d'azione (visibilità):**
  - **è la parte di programma in cui la variabile è nota (visibile) e può essere manipolata**

  - **I due concetti sono legati**

Ciclo di vita di un identificatore

- **Quando un identificatore viene dichiarato, si crea una associazione tra il simbolo e l'informazione legata all'oggetto che denota.**
- **Questo legame rimane durante tutta l'attivazione del programma**

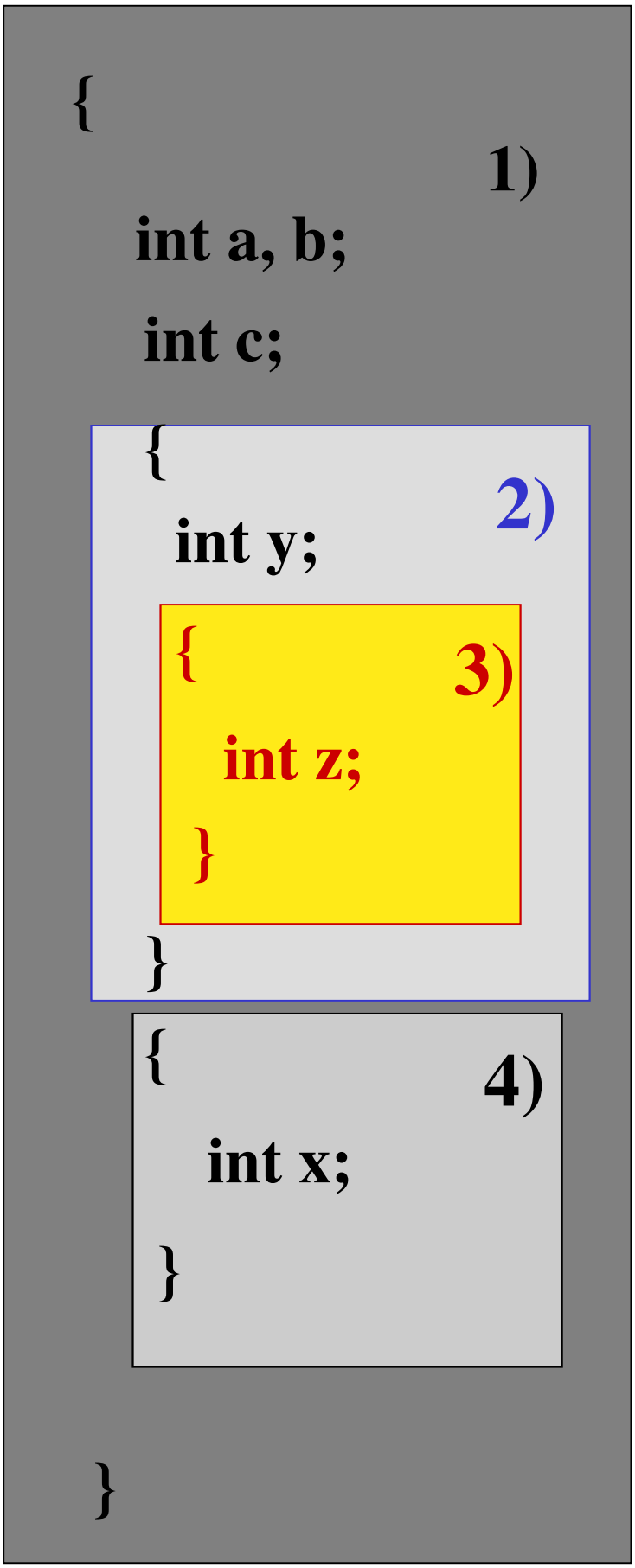
Variabile A di tipo T associata alla locazione I



Funzione F associata alla locazione J dove è memorizzata la funzione

Ciclo di vita di un identificatore

- Tali informazioni permangono per il ciclo di vita dell'identificatore
  - **Il CICLO DI VITA corrisponde alla durata dell'attivazione del blocco di programma in cui le dichiarazioni compaiono**
  - **Il legame si attiva all'entrata del blocco e si scioglie all'uscita**
  - **L'identificatore nasce e muore con il blocco**



a, b, c sono attivi in tutto il blocco 1 (ma anche in 2, 3 e 4)

y è attivo in 2 (ma anche in 3)

z è attivo in 3

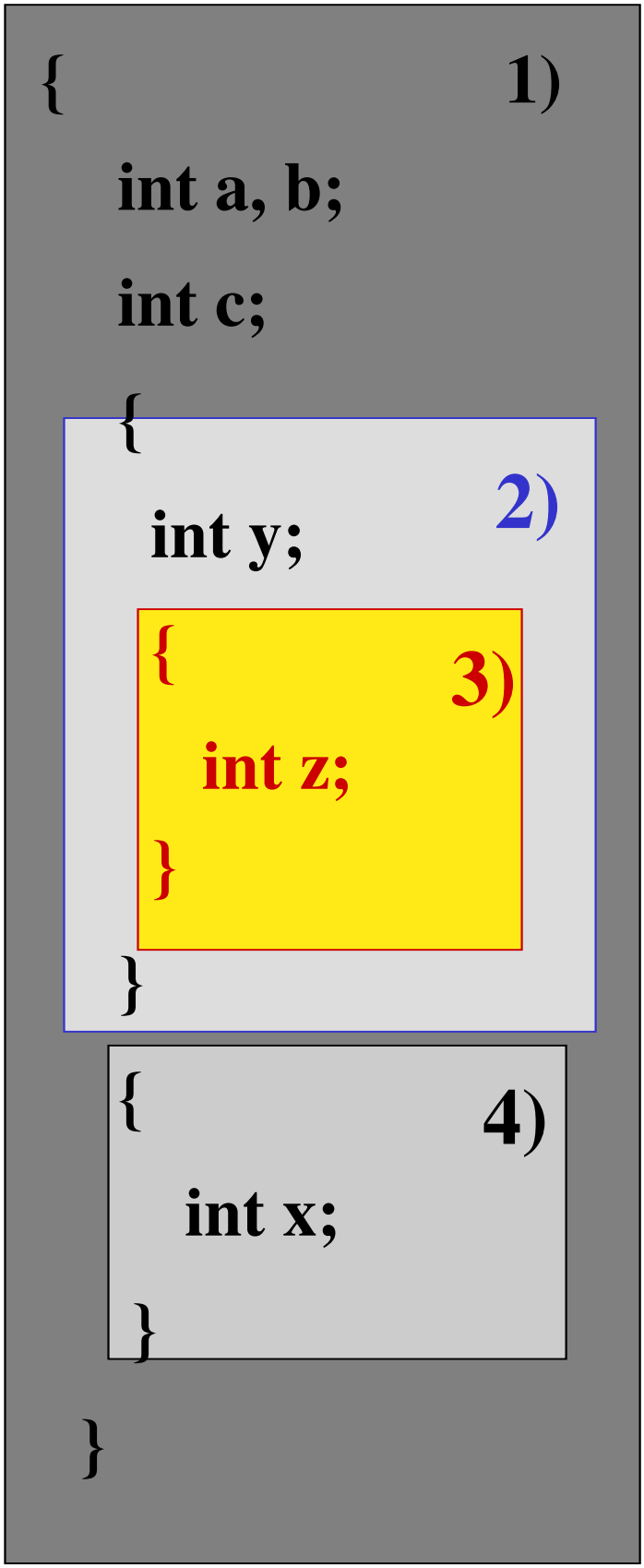
x è attivo in 4

Campo d'azione (scope)

- **Il campo d'azione di un identificatore definisce l'insieme delle porzioni di programma che possono utilizzare (e che quindi vedono) l'identificatore dichiarato**
- **In un blocco possono essere usati solo gli identificatori visibili in quel blocco**
- **Come sono visti gli identificatori dichiarati in un blocco negli altri blocchi?**
  - **Bisogna definire delle regole di visibilità, ossia le regole di comunicazione tra blocchi**

Regole di visibilità

- *Gli identificatori sono accessibili:*
  - *solo all'interno del blocco in cui sono dichiarati*
  - *da tutti i blocchi contenuti in esso*



a, b, c sono visibili in 1, 2, 3, 4

y in 2, 3

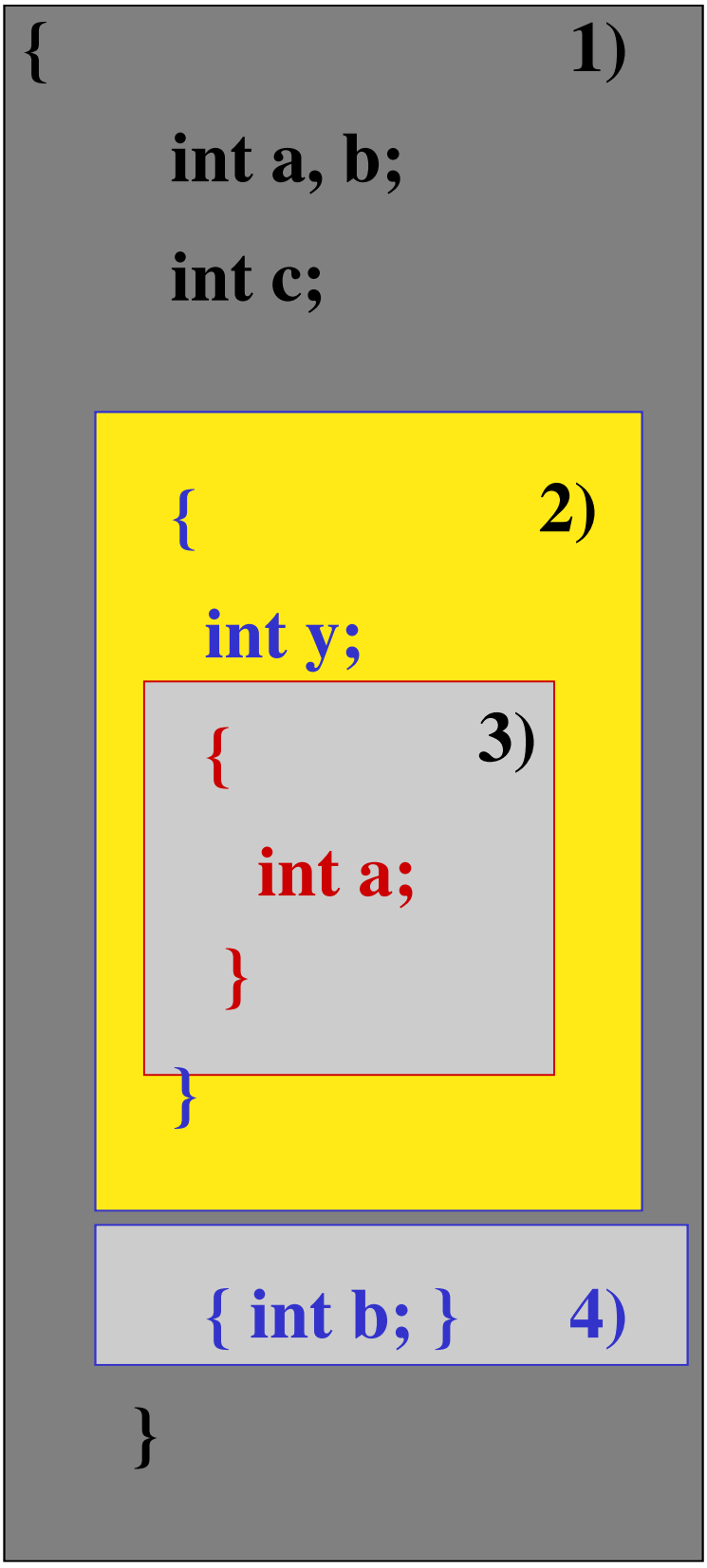
z in 3

x in 4

Regole di visibilità

- *Gli identificatori sono accessibili:*
  - *solo all'interno del blocco in cui sono dichiarati*
  - *da tutti i blocchi in esso contenuti*
    - *A meno che lo stesso identificatore non sia dichiarato di nuovo all'interno di un blocco più interno*
      - **Questa una nuova dichiarazione maschera la precedente e in quel blocco interno ha effetto l'ultima dichiarazione**

Esempio



a, b, c sono visibili in
1, 2
b, c in 3
a, c in 4

y in 2, 3

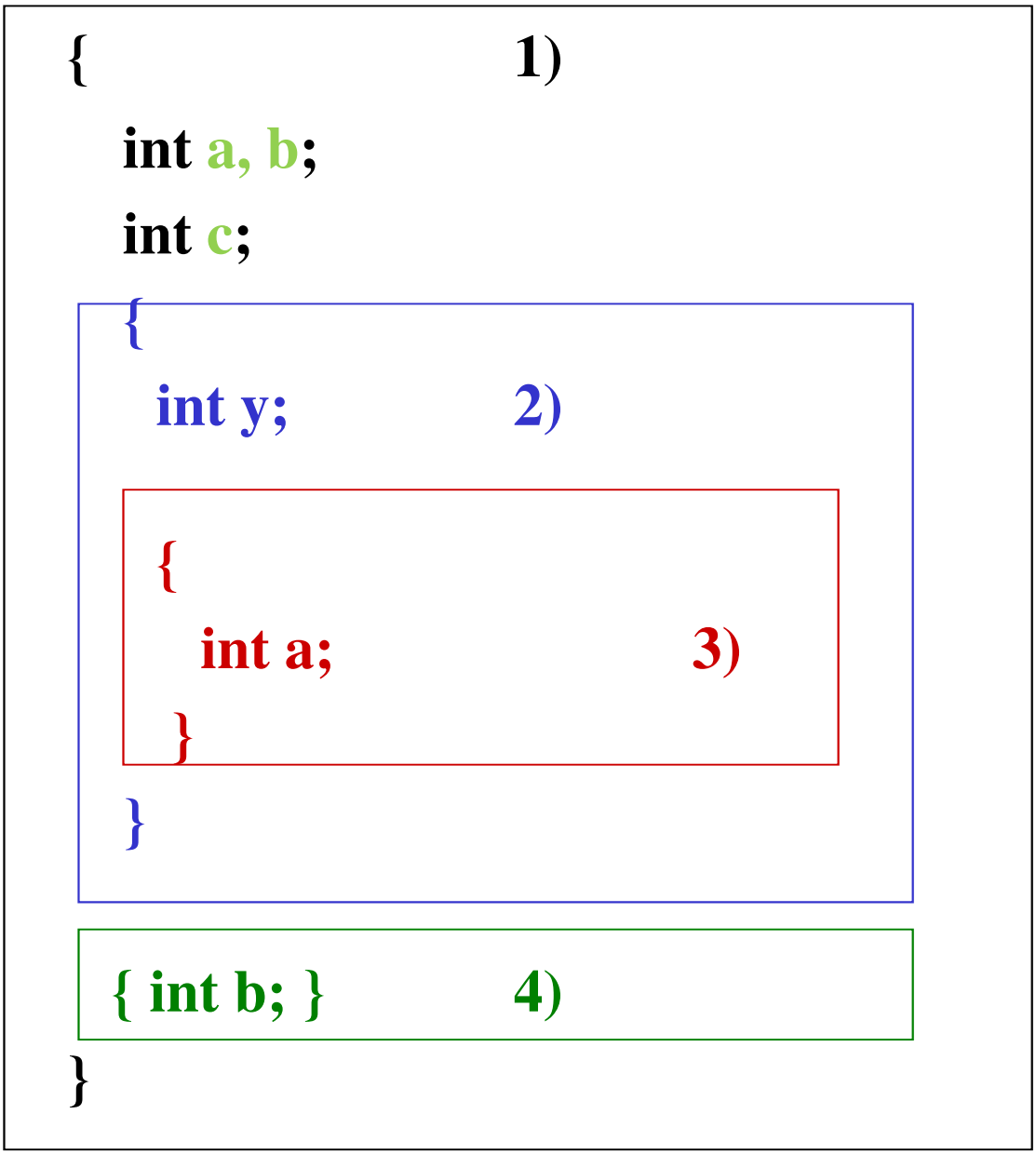
a in 3

b in 4

Campo d'azione (scope)

- **Un blocco**
  - **Permette l'allocazione di memoria per le variabili solo quando necessaria**
  - **Se la memoria è scarsa, l'uscita da un blocco libera la memoria per le variabili usate all'interno del blocco, lasciandola libera per altri scopi**

  - **Questo perché il ciclo di vita di un identificatore coincide con la durata dell'attivazione del blocco in cui è definita**



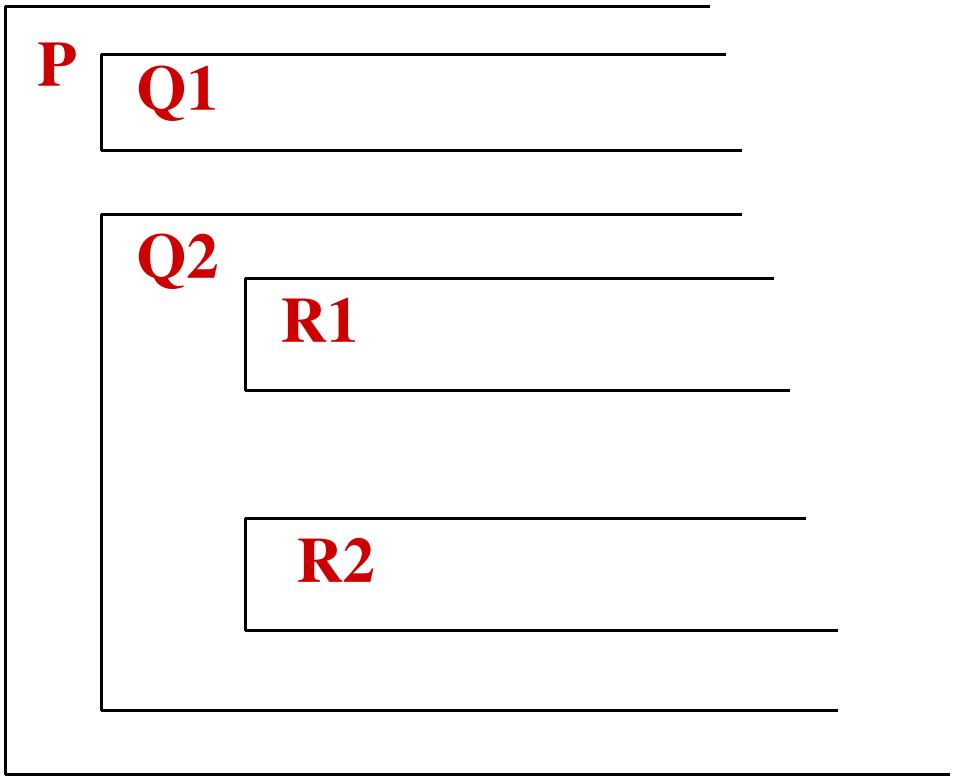
- Quando entriamo nel blocco 2), 1) è ancora attivo quindi vediamo a, b, c.
- Quando entriamo in 3), la nuova definizione di a maschera quella in 1)
- In 4) la definizione di b maschera quella in 1)

```
{
   int a = 2;
   printf ("%d\n", a);     /* stampa 2 */
   {              /* inizio blocco interno */
        int a =5;
         printf ("%d\n", a);
                   /* stampa 5 */
   }
   printf ("%d\n", ++a);
                  /* stampa  3 */
}
```

- Le due *a* sono visibili ciascuno all'interno del proprio blocco, non sono passate da un blocco all'altra

- **Un nome in un blocco esterno è valido in un blocco interno a meno che non sia stato ridefinito**

```
{
   int a = 2;
   printf ("%d\n", a);     /* stampa 2 */
   {              /* inizio blocco interno */
        int b =6;
         printf ("%d\n", a*b);
                  /* stampa 12 */
   }
   printf ("%d\n", ++a);
                  /* stampa  3 */
}
```



- Tutti gli identificatori dichiarati in P sono visibili ovunque
- Gli identificatori dichiarati in Q2 sono visibili in R1 e R2
  - *(a meno di mascheramenti)*
- Tra Q1 e Q2 non c'è visibilità
- Tra R1 e R2 non c'è visibilità
- Gli identificatori in R1 e R2 sono visibili solo al loro interno

Formattazione dell'output

- Con printf può essere ottenuta una precisa formattazione dell'output.
- Ogni invocazione di printf contiene una *stringa di controllo del formato che descrive appunto il formato dell'output.*
- La stringa di controllo del formato è composta da *indicatori di conversione, flag, dimensioni di campo, precisioni e caratteri letterali. Uniti al segno di percentuale (%), questi formano le specifiche di conversione.*

Formattazione dell'output

- *La funzione printf* ha le seguenti capacità di formattazione:

1. *Arrotondamento dei valori in virgola mobile al numero indicato di cifre decimali.*
2. *Allineamento di una colonna di numeri con i separatori dei decimali che compaiono uno* sull'altro.
3. *Allineamento a destra e a sinistra degli output.*
4. *Inserimento di caratteri letterali in posizioni precise della riga inviata in output.*
5. Rappresentazione dei numeri in virgola mobile in formato esponenziale.
6. Rappresentazione degli interi senza segno in formato ottale ed esadecimale.
7. Visualizzazione di tutti i tipi di dato in campi di dimensione e precisione prefissate.

Formattazione dell'output

- La funzione printf ha la forma:

  printf (stringa di controllo del formato, altri argomenti);

- La ***stringa di controllo del formato*** *descrive il formato dell'output, mentre gli*
- ***altri argomenti*** (che sono opzionali) corrispondono alle singole specifiche di conversione inserite nella *stringa di controllo del formato.*
- *Ogni specifica di conversione incomincia con un segno di* percentuale (**%**) e termina con un indicatore di conversione.
- In una stringa di controllo del formato potranno essere inserite molte specifiche di conversione.

Visualizzare gli interi

- Un intero è un numero, come 776, 0 o –52, che non contiene virgole decimali.
- I valori interi possono essere visualizzati in molti formati.

Indicatore di conversione	Descrizione
`d`	Visualizza un intero decimale con segno.
`i`	Visualizza un intero decimale con segno. [*Nota*: il comportamento degli indicatori `i` e `d` sarà diverso quando saranno utilizzati con `scanf`].
`o`	Visualizza un intero ottale senza segno.
`u`	Visualizza un intero decimale senza segno.
`x` o `X`	Visualizza un intero esadecimale senza segno. `X` induce la visualizzazione delle cifre `0-9` e delle lettere `A-F`, mentre `x` induce la visualizzazione delle cifre `0-9` e delle lettere `a-f`.
`h` o `l` (lettera `l`)	Posto dinanzi a ogni indicatore di conversione per gli interi, per indicare che sarà visualizzato rispettivamente un intero `short` o `long`. Più precisamente le lettere `h` e `l` sono chiamate *modificatori di lunghezza*.

Visualizzare gli interi

```
1 /* esempio */
2 Usare gli indicatori di conversione per gli interi */
3 #include <stdio.h>
4
5 int main()
6 {
7 printf("%d\n", 455);
8 printf("%i\n", 455); /* i è come d
9 printf("%d\n", +455);
10 printf(„%d\n", –455);
11 printf(„%hd\n", 32000);
12 printf(„%ld\n", 2000000000);
13 printf("%o\n", 455);
14 printf("%u\n", 455);
15 printf("%u\n", –455);
16 printf("%x\n", 455);
17 printf("%X\n", 455);
18
19 return 0; /* indica che il programma è terminato con
     successo */
20
21 } /* fine della funzione main */
```

```
455
455
455
–455
32000
2000000000
707
455
4294966841
1c7
1C7
```

Visualizzare i numeri in virgola mobile

- Un valore in virgola mobile contiene una virgola decimale come 33,5, 0,0 o 657,983.
- I valori in virgola mobile potranno essere visualizzati in molti formati.

Indicatore di conversione	Descrizione
`e` o `E`	Visualizza un valore in virgola mobile nella notazione esponenziale.
`f`	Visualizza un valore in virgola mobile nella notazione in virgola fissa.
`g` o `G`	Visualizza un valore in virgola mobile sia nel formato di `f` che in quello di `e` (o di `E`) in base alla grandezza del valore.
`L`	Posto dinanzi a un qualsiasi indicatore di conversione per i numeri in virgola mobile, per indicare che sarà visualizzato un valore in virgola mobile `long double`.

Visualizzare i numeri in virgola mobile

```
1 /* esempio */
2 /* Visualizzare i numeri in virgola mobile con gli
     indicatori
3 di conversione per i valori in virgola mobile */
4
5 #include <stdio.h>
6
7 int main()
8 {
9 printf("%e\n", 1234567.89);
10 printf("%e\n", +1234567.89);
11 printf("%e\n", -1234567.89);
12 printf("%E\n", 1234567.89);
13 printf("%f\n", 1234567.89);
14 printf("%g\n", 1234567.89);
15 printf("%G\n", 1234567.89);
16
17 return 0; /* indica che il programma è terminato con
     successo */
18
19 } /* fine della funzione main */
```

```
1.234568e+06
1.234568e+06
–1.234568e+06
1.234568E+06
1234567.890000
1.23457e+06
1.23457E+06
```

Visualizzazione di stringhe e caratteri

```
1 /* esempio */
2 /* Visualizzare stringhe e caratteri */
3 #include <stdio.h>
4
5 int main()
6 {
7 char character = 'A'; /* inizia
    */
8 char string[] = "This is a string"; /* inizializza
il vettore di elementi di tipo char */
9 const char *stringPtr = "This is also a string";
*/ puntatore a char */
10
11 printf("%c\n", character);
12 printf("%s\n", "This is a string");
13 printf("%s\n", string);
14 printf("%s\n", stringPtr);
15
16 return 0; /* indica che il programma è terminato con
    successo */
17
18 } /* fine della funzione main */
```

A
This is a string
This is a string
This is also a string

Altri operatori

Indicatore di conversione	Descrizione
p	Visualizza il valore di un puntatore in un formato definito dall'implementazione.
n	Immagazzina il numero di caratteri già inviato in output dall'istruzione `printf` corrente. Come argomento corrispondente dovrà essere fornito un puntatore a un intero. L'indicatore di conversione non visualizzerà niente.
%	Visualizza il simbolo di percentuale.

- L'indicatore di conversione **n** immagazzina il numero di caratteri già inviati in output dall'istruzione printf corrente: infatti, l'argomento corrispondente è un puntatore alla variabile intera nella quale sarà immagazzinato il valore.
- La specifica di conversione **%n** non visualizza niente.
- L'indicatore di conversione **%** visualizza un segno di percentuale.

Altri operatori

```
1 /* esempio
2 /* Usare
3 #include
4
5 int main()
6 {
7 int *ptr; /*
8 int x = 12345; /* inizializza la variabile x di tipo int
9 int y; /* dichiara la variabile y di tipo int */
10
11 ptr = &x; /* assegna l'indirizzo di x a ptr */
12 printf("The value of ptr is %p\n", ptr);
13 printf("The address of x is %p\n\n", &x);
14
15 printf("Total characters printed on this line is:%n", &y);
16 printf(" %d\n\n", y);
17
18 y = printf("This line has 28 characters\n");
19 printf("%d characters were printed\n\n", y);
20
21 printf("Printing a %% in a format control string\n");
22
23 return 0; /* indica che il programma è terminato con successo */
24
25 } /* fine della funzione main */
```

```
The value of ptr is 0012FF78
The address of x is 0012FF78
Total characters printed on this line is: 38
This line has 28 characters
28 characters were printed
Printing a % in a format control string
```

Visualizzare con le dimensioni di campo e le precisioni

- L'esatta misura del campo in cui saranno visualizzati i dati è specificata dalla cosiddetta *dimensione di campo.*
- *Nel caso che la dimensione del campo sia maggiore del dato da visualizzare* questo, di norma, sarà allineato a destra all'interno di quel campo.
- Normalmente l'intero che rappresenta la dimensione del campo è inserito tra il segno di percentuale (**%**) e l'indicatore di conversione (ad esempio, **%4d**).
- Il programma che segue visualizzerà due gruppi di cinque numeri, allineando a destra quelli che contengono meno cifre della dimensione del Campo.
- Osservate che, per visualizzare dei valori con una dimensione maggiore di quella del campo, questa sarà incrementata, e che il segno dei valori negativi utilizzerà una posizione all'interno della stessa.

Visualizzare con le dimensioni di campo e le precisioni

```
1 /* esempio */
2 /* Visualizzare gli interi allineati a destra */
3 #include <stdio.h>
4
5 int main()
6 {
7 printf("%4d\n", 1);
8 printf("%4d\n", 12);
9 printf("%4d\n", 123);
10 printf("%4d\n", 1234);
11 printf("%4d\n\n", 12345);
12
13 printf("%4d\n", –1);
14 printf("%4d\n", –12);
15 printf("%4d\n", –123);
16 printf("%4d\n", -1234);
17 printf("%4d\n", -12345);
18
19 return 0; /* indica che il programma è terminato con
     successo */
20
21 } /* fine della funzione main */
```

```
   1
  12
 123
1234
12345

  –1
 –12
–123
–1234
–12345
```

Specifiche di precisione

- La funzione printf fornisce anche la possibilità di specificare la *precisione con cui il dato* dovrà essere visualizzato.
- La precisione ha un significato diverso per i vari tipi di dato.
- Utilizzata insieme agli indicatori di conversione per gli interi, la precisione indica **il numero minimo di cifre da visualizzare.**
- Nel caso che il valore visualizzato contenga meno cifre di quelle indicate dalla precisione, davanti al suddetto valore saranno inseriti degli zeri finché il numero totale delle cifre non risulti equivalente a quello della precisione.
- **La precisione predefinita per gli interi è 1**.
- Utilizzata con gli indicatori di conversione **e, E e f** per i valori in virgola mobile, **la precisione indica il numero di cifre che dovranno comparire dopo la virgola dei decimali**.

Specifiche di precisione

- Utilizzata con gli indicatori di conversione **g e G**, la precisione indica **il numero massimo di cifre significative da visualizzare**.
- Utilizzata con l'indicatore di conversione **s**, la precisione indica il numero massimo di caratteri della stringa che dovranno essere scritti.
- **Per utilizzare la precisione, inserirete un punto (.) seguito dall'intero che rappresenta la precisione, tra il segno di percentuale e l'indicatore di conversione.**
- A seguire verrà mostrato l'utilizzo della precisione nelle stringhe di controllo del formato.
- Osservate che un valore in virgola mobile sarà arrotondato, qualora sia visualizzato con una precisione inferiore al numero originale delle sue cifre decimali.

Specifiche di precisione

```
1 /* Esempio */
2 /* Usare la precisione
3 i numeri in virgola m
4 #include <stdio.h>
5
6 int main()
7 {
8 int i = 873; /* iniziali
9 float f = 123.94536; /
di tipo double */
10 char s[] = “Happy B
di caratteri s */
11
12 printf(“Using precision for integers\n”);
13 printf(“\t%.4d\n\t%.9d\n\n”, i, i);
14
15 printf(“Using precision for floating-point numbers\n”);
16 printf(“\t%.3f\n\t%.3e\n\t%.3g\n\n”, f, f, f);
17
18 printf(„Using precision for strings\n”);
19 printf(“\t%.11s\n”, s);
20
21 return 0; /* indica che il programma è terminato con successo */
22
23 } /* fine della funzione main */
```

```
Using precision for integers
0873
000000873
Using precision for floating-
point numbers
123.945
1.239e+002
124
Using precision for strings
Happy Birth
```

Array

Array

- Un array (o vettore) è un tipo di dato composto (aggregato) usato per rappresentare un gruppo di variabili dello stesso tipo
  - tutte le variabili hanno lo stesso nome e lo stesso tipo
  - sono differenziate da un indice che indica la loro posizione
  - le variabili sono memorizzate consecutivamente
- Per dichiarare un array si deve specificare:
  - tipo di dato, nome e dimensione

```
<tipo>
   <nome_array>[<dimensione_array>]

int voti[25];
char cognome[21]
```

(Vettori) Array

- int c[8]
- c[1], c[2], …
  - int tipo elementi dell'array
  - c nome array (identificatore valido)
  - 8 numero degli elementi dell'array
  - c[i] elemento di un array

c[0]	12
c[1]	-1
c[2]	0
c[3]	67
c[4]	5
c[5]	93
c[6]	121
c[7]	-90

c nome array
[i] indice

c[i] elemento i-esimo
(i+1)-esimo

- Il numero di elementi deve essere un intero (o un'espressione intera)
- Il primo elemento ha SEMPRE l'indice pari a 0

Array

```
int d_temp[365];
float av_temp[7];
double av_rate[ ];
```

- Se non si specifica il numero di elementi, viene dichiarato un *unsized* array, e la dimensione sarà calcolata quando si inseriscono i valori
- Ogni singolo elemento è individuato da un indice

```
d_temp[4]
```

- È possibile assegnare un valore ad ogni singolo elemento

```
d_temp[4] = 32;
```

- Il primo elemento ha indice 0

```
d_temp[0]
```

- L'ultimo indice è pari alla dimensione meno uno

```
d_temp[364]
```

Array

- **dichiarazioni**

```
int d_temp[365];
float av_temp[7];
double av_rate[50];
```

- **elementi**

```
d_temp[4]
float av_temp[6]
double av_rate[i]
```

- **Notare la differenza tra la dichiarazione e il singolo elemento**
  - **La dichiarazione compare solo all'inizio di blocchi funzionali nelle parti dedicate alle dichiarazioni**
  - **Un elemento può apparire in un'espressione, nella parte destra o sinistra di un'assegnazione**

Inizializzazione degli array

- I valori degli elementi degli array devono essere inizializzati per evitare che il loro valore sia indefinito
- Possono essere inizializzati in diversi modi
  - *in fase di dichiarazione*

    `int ar[5] = {1, 2, 3, 4, 5};`
    - ar[0] =1, ar[1] =2, ar[2]=3, ar[3] =4, ar[4] =5

    `int ar[8] = {1, 2, 3, 4, 5};`
    - ar[0] =1, ar[1] =2, ar[2]=3, ar[3] =4, ar[4] =5, ar[5] =0, ar[6] =0, ar[7] =0

    `int ar[25] = {0} ;`
    - inizializza a zero tutti gli elementi

    `int ar [ ] = {0, 1, 2, 3, 4}`
    - crea un array di cinque elementi

    `int ar[2] = {1, 2, 3};`
    - errato. 2 elementi e 3 valori

Inizializzazione degli array (2)

- *con assegnazione ai singoli elementi*

```
int array2[4];
array2[0]=3;
array2[1]=6;
array2[2]=13;
array2[3]=9;
```

- *con un ciclo, per esempio con il for*

```
int array3[10];
int i;
for (i=0; i< 10; i++)
      array3[i] = i+1;
```

Inizializzazione degli array (3): *Uso della scanf*

```
#include <stdio.h>
int main(void) {
   int cinque[5];
   int i;
   printf(”Inserire 5 interi ");

   for(i=0 ; i<5 ; i++) {
     scanf("%d",&cinque[i]);
   }

   for(i=0 ; i<5 ; i++) {
    printf("cinque[%d] = %d\n",
            i, cinque[i]);}
   return 0;}
```

leggi_array.c

Stampa degli array

- Il modo più comune è usare un ciclo, per esempio un for

```
#include <stdio.h>
int main(int) {
   int cinque[5] =
      {1,2,3,4,5};
   int i;
   for(i=0 ; i<6 ; i++) {
     printf("cinque[%d]=%d\n",
       i, cinque[i]);
   }
   return 0;
}
```

stampa_array.c

Stampa degli array

- **Notate!!!**

```
Output:
cinque[0] =  1
cinque[1] =  2
cinque[2] =  3
cinque[3] =  4
cinque[4] =  5
cinque [5] = 6684216
```

```c
#include <stdio.h>
#define N 5

int main (void) {
   int ar[N]= {0}, i;
   ar[1] = 5;
   ar[2] = 3;
   ar[N-1] = ar[1]*ar[2];

   printf("%10s %10s \n",
         "Indice", "Valore");
   for (i=0; i <= (N-1); i++)
    printf("%d %d\n",i,ar[i]);

   return 0;
 }
```

```c
#include <stdio.h>
#define N 7

int main (void) {

   int ar[N], i, base=2;
   printf("%10s %10s \n",
          "Indice", "Valore");
   for (i=0; i <= (N-1); i++)
      {
        ar[i] = 2*i +2;
        printf("%10d  %10d\n",
               i, ar[i]); }
   return 0;
}
```

array.c

```c
#include <stdio.h>
#define SIZE 10
void leggi (int [], int);
int main (void) {
  int vet[SIZE], i;
  leggi(vet, SIZE);
  for (i=0; i <= (SIZE-1); i++)
     printf("%10d  %10d\n", i,
           vet[i]);
return 0;
}

void leggi (int vet[], int v) {
  int i;
  printf("Introduci i valori
  dell'array %d\n”);
  for (i=0; i<= v-1; ++i) {
      printf("valore %d\n", i);
      scanf("%d", &vet[i]);
           }
  }
```

array2.c