

Elementi Di Informatica E Programmazione

Prof. Andrea Loreggia



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Il concetto di funzione
- Parametri formali e attuali
- Il valore di ritorno
- Definizione e chiamata di funzioni
- Passaggio dei parametri
- Corpo della funzione

- Riuso di codice esistente
 - Funzionalità simili in programmi diversi
 - Funzionalità ripetute all'interno dello stesso programma
 - Minore tempo di sviluppo
 - Frammenti di codice già verificati
 - Utilizzo di parti di codice scritte da altri
 - Funzioni di libreria
 - Sviluppo collaborativo

Come riusare il codice? (1/3)



➤ Copia-e-incolla

- Semplice, ma poco efficace
- Occorre adattare il codice incollato, ritoccando i nomi delle variabili e costanti utilizzati
- Se si scopre un errore, occorre correggerlo in tutti i punti in cui è stato incollato
- Nel listato finale non è evidente che si sta riutilizzando la stessa funzionalità
- Occorre disporre del codice originario
- Occorre capire il codice originario

Come riusare il codice? (2/3)

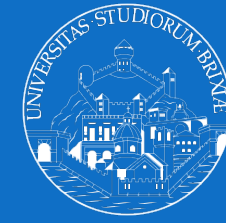


UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

➤ Definizione di funzioni

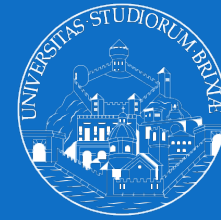
- Dichiarazione esplicita che una certa funzionalità viene utilizzata in più punti
- Si separa la definizione della funzionalità rispetto al punto in cui questa viene utilizzata
- La stessa funzione può essere usata più volte con parametri diversi

Come riusare il codice? (3/3)



- Ogni miglioramento o correzione è automaticamente disponibile in tutti i punti in cui la funzione viene usata
- Nel listato finale è evidente che si sta riutilizzando la stessa funzionalità
- Non occorre disporre del codice originario Non occorre capire il codice originario

Principio di funzionamento (1/3)



```
int main(void)
{
    int x, y ;

    /* leggi un numero
       tra 50 e 100 e
       memorizzalo
       in x */
    /* leggi un numero
       tra 1 e 10 e
       memorizzalo
       in y */

    printf("%d %d\n",
           x, y ) ;
}
```

Principio di funzionamento (2/3)

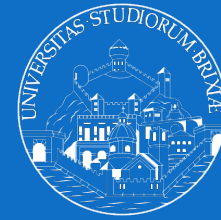


```
int main(void)
{
    int x, y ;

    x = leggi(50, 100) ;
    y = leggi(1, 10) ;

    printf("%d %d\n",
           x, y ) ;
}
```


Principio di funzionamento (3/3)



```
int main(void)
{
    int x, y ;

    x = leggi(50, 100) ;
    y = leggi(1, 10) ;

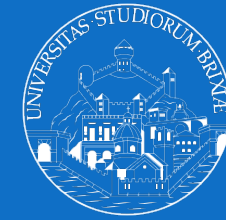
    printf("%d %d\n",
           x, y ) ;
}
```

```
int leggi(int min,
           int max)
{
    int v ;

    do {
        scanf("%d", &v) ;
    } while( v<min ||
            v>max) ;

    return v ;
}
```

Principio di funzionamento (3/3)



```
int main(void)
{
    int x, y ;
    x = leggi(50, 100) ;
    y = leggi(1, 10) ;
    printf("%d %d\n",
           x, y ) ;
}
```

min=50

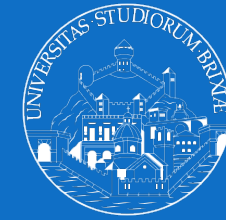
max=100

```
int leggi(int min,
          int max)
{
    int v ;

    do {
        scanf("%d", &v) ;
    } while( v<min ||
            v>max) ;

    return v ;
}
```

Principio di funzionamento (3/3)



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

```
int main(void)
{
    int x, y ;
    x = leggi(50, 100) ;
    y = leggi(1, 10) ;
    printf("%d %d\n",
        x, y ) ;
}
```

Chiamante

min=50

max=100

```
int leggi(int min,
          int max)
{
    int v ;

    do {
        scanf("%d", &v) ;
    } while( v<min ||
            v>max) ;

    return v ;
}
```

Chiamato

Principio di funzionamento (3/3)



```
int main(void)
{
    int x, y ;

    x = leggi(50, 100) ;
    y = leggi(1, 10) ;

    printf("%d %d\n",
           x, y ) ;
}
```

min=1

max=10

```
int leggi(int min,
          int max)
{
    int v ;

    do {
        scanf("%d", &v) ;
    } while( v<min ||
            v>max) ;

    return v ;
}
```

- La definizione di una **funzione** delimita un frammento di codice riutilizzabile più volte
- La funzione può essere **chiamata** più volte
- Può ricevere dei **parametri** diversi in ogni chiamata
- Può restituire un **valore di ritorno** al chiamante
 - Istruzione **return**

Miglioramento della funzione



```
int leggi(int min, int max)
{
    char riga[80] ;
    int v ;

    do {
        gets(riga) ;
        v = atoi(riga) ;
        if(v<min) printf("Piccolo: min %d\n", min) ;
        if(v>max) printf("Grande: max %d\n", max) ;
    } while( v<min || v>max) ;

    return v ;
}
```

Parametri di una funzione



- Le funzioni possono ricevere dei parametri dal proprio chiamante
- Nella funzione:
 - Parametri formali
 - Nomi "interni" dei parametri

```
int leggi(int min,  
          int max)  
{  
    ...  
}
```

Parametri di una funzione

- Le funzioni possono ricevere dei parametri dal proprio chiamante

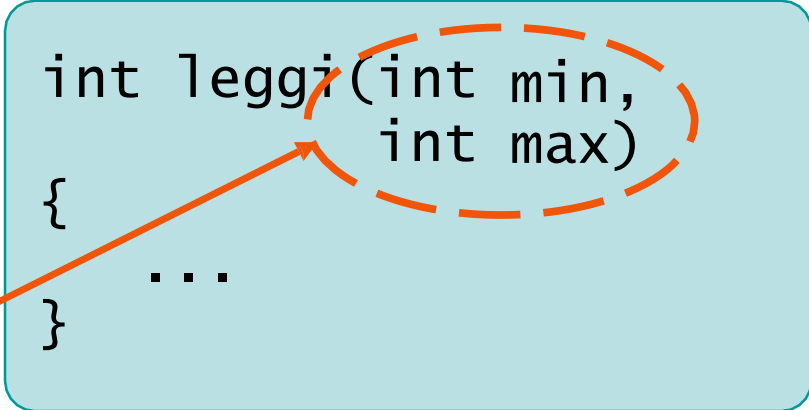
- Nella funzione:

- Parametri formali
- Nomi "interni" dei parametri

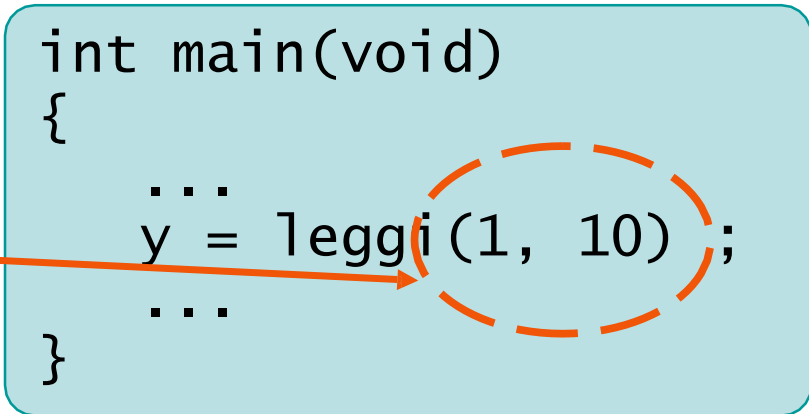
- Nel chiamante:

- Parametri attuali
- Valori effettivi (costanti, variabili, espressioni)

```
int leggi(int min,  
          int max)  
{  
    ...  
}
```



```
int main(void)  
{  
    ...  
    y = leggi(1, 10);  
    ...  
}
```



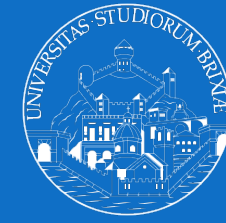
Parametri formali (1/2)

- Uno o più parametri
- Tipo del parametro
 - Tipo scalare
 - Vettore o matrice
- Nome del parametro
- Nel caso in cui la funzione non abbia bisogno di parametri, si usa la parola chiave `void`

```
int leggi(int min,  
          int max)  
{  
    ...  
}
```

```
int stampa_menu(void)  
{  
    ...  
}
```

Parametri formali (2/2)



➤ Per parametri vettoriali esistono 3 sintassi alternative

- `int v[]`
- `int v[MAX]`
- `int *v`

```
int leggi(int v[])  
{  
    ...  
}
```

➤ Per parametri matriciali

- `int m[RIGHE][COL]`
- `int m[][COL]`

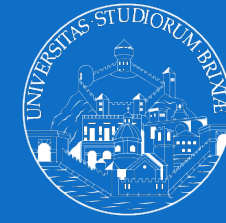
```
int sudoku(int m[9][9])  
{  
    ...  
}
```

- Il valore della dimensione del vettore (es. MAX)
 - Viene totalmente ignorato dal meccanismo di chiamata
 - Non sarebbe comunque disponibile alla funzione chiamata
 - Meglio per chiarezza ometterlo
 - Si suggerisce di passare un ulteriore parametro contenente l'occupazione del vettore

► Nel caso di matrici

- Il secondo parametro (es. COL) è obbligatorio e deve essere una costante
- Il primo parametro viene ignorato e può essere omissso
- Per matrici pluri-dimensionali, occorre specificare tutti i parametri tranne il primo

Parametri attuali (1/2)



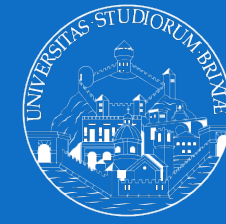
- Uno o più valori, in esatta corrispondenza con i parametri formali
- Tipi di dato compatibili con i parametri formali
- È possibile usare
 - Costanti
 - Variabili
 - Espressioni

```
int main(void)
{
    y = leggi(1, 10);
}
```

```
int main(void)
{
    y = leggi(a, b);
}
```

```
int main(void)
{
    y = leggi(a+b+1,
              c*2);
}
```

Parametri attuali (2/2)



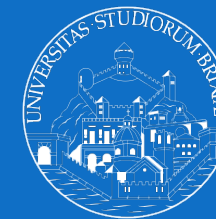
- I nomi delle variabili usate non hanno alcuna relazione con i nomi dei parametri formali

```
int main(void)
{
    ...
    y = leggi(a, b) ;
    ...
    min=a
    max=b
}
```

- Le parentesi sono sempre necessarie, anche se non vi sono parametri

```
int main(void)
{
    ...
    y = stampa_menu() ;
    ...
}
```

Valore di ritorno



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Ogni funzione può ritornare un valore al proprio chiamante
- Il tipo del valore di ritorno deve essere **scalare**
- L'istruzione **return**
 - Termina l'esecuzione della funzione
 - Rende disponibile il valore al chiamante

```
int leggi(int min,  
          int max)  
{  
    int v ;  
    scanf("%d", &v) ;  
    return v ;  
}
```

```
int main(void)  
{  
    ...  
    y = leggi(a, b) ;  
    ...  
}
```

Tipo del valore di ritorno



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Valore scalare
 - `char`, `int` (o varianti), `float`, `double`
- Tipi avanzati
 - Puntatori, `struct`
- Nessun valore ritornato
 - `void`

```
double sqrt(double a)
{
    ...
}
```

```
void stampa_err(int x)
{
    ...
}
```


L'istruzione return (1/2)



- Restituisce il valore
 - Costante
 - Variabile
 - Espressione
- Il tipo deve essere compatibile con il tipo dichiarato per il valore di ritorno
- Sintassi
 - `return x ;`
 - `return(x) ;`
- L'esecuzione della funzione viene interrotta

L'istruzione return (2/2)



- Per funzioni che non ritornano valori (`void`):
 - `return ;`
- Il raggiungimento della fine del corpo della funzione `}` equivale ad un'istruzione `return` senza parametri
 - Permesso solo per funzioni `void`
- Per funzioni non-`void`, è obbligatorio che la funzione ritorni **sempre** un valore

Nel chiamante...



- Il chiamante può:
 - Ignorare il valore ritornato
 - `scanf("%d", &x) ;`
 - Memorizzarlo in una variabile
 - `y = sin(x) ;`
 - Utilizzarlo in un'espressione
 - `if (sqrt(x*x+y*y)>z) ...`

- Le funzioni di tipo matematico ritornano sempre un valore double
 - Le funzioni che non devono calcolare un valore (ma effettuare delle operazioni, per esempio) ritornano solitamente un valore int
 - Valore di ritorno `== 0` \Rightarrow tutto ok
 - Valore di ritorno `!= 0` \Rightarrow si è verificato un errore
- Molte eccezioni importanti: `strcmp`, `scanf`, ...

- Il linguaggio C prevede 3 distinti momenti :
 - La dichiarazione (prototipo o function prototype)
 - L'interfaccia della funzione
 - Solitamente: prima del `main()`
 - La definizione
 - L'implementazione della funzione
 - Solitamente: al fondo del file, dopo il `main()`
 - La chiamata
 - L'utilizzo della funzione
 - Solitamente: dentro il corpo del `main()` o di altre funzioni

Dichiarazione o prototipo



```
int leggi(int min, int max) ;
```

Tipo del
valore di
ritorno

Nome della
funzione

Tipi e nomi
dei parametri
formali

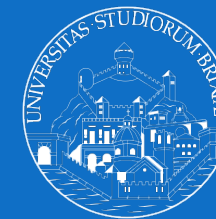
Punto-e-
virgola

Scopo del prototipo



- Dichiarare che esiste una funzione con il nome dato, e dichiarare i tipi dei parametri formali e del valore di ritorno
- Dal prototipo in avanti, si può chiamare tale funzione dal corpo di qualsiasi funzione (compreso il `main`)
- Non è errore se la funzione non viene chiamata
- I file `.h` contengono centinaia di prototipi delle funzioni di libreria
- I prototipi sono posti solitamente ad inizio file

Definizione o implementazione



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Tipo del
valore di
ritorno

Nome della
funzione

Tipi e nomi
dei parametri
formali

```
int leggi(int min, int max)
{
    ... codice della funzione ...
}
```

Corpo della funzione
{ ... }

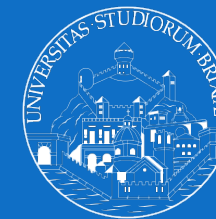
Nessun
punto-e-virgola

Scopo della definizione



- Contiene il codice vero e proprio della funzione
- Può essere posizionata ovunque nel file (al di fuori del corpo di altre funzioni)
- Il nome della funzione ed i tipi dei parametri e del valore di ritorno devono coincidere con quanto dichiarato nel prototipo
- Il corpo della funzione può essere arbitrariamente complesso, e si possono chiamare altre funzioni

Chiamata o invocazione



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Funzione
chiamante

Valori dei parametri
attuali

```
int main(void)
{
    int x, a, b ;
    ...
    x = leggi(a, b) ;
    ..
}
```

Uso del valore di
ritorno

Chiamata della
funzione

- Le espressioni corrispondenti ai parametri attuali vengono valutate (e ne viene calcolato il valore numerico)
 - **Compaiono le variabili del chiamante**
- I valori dei parametri attuali vengono copiati nei parametri formali
- La funzione viene eseguita
- All'istruzione `return`, il flusso di esecuzione torna al chiamante
- Il valore di ritorno viene usato o memorizzato

Riassumendo...

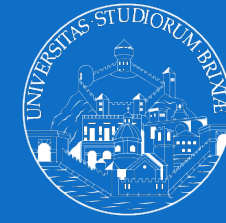


```
int leggi(int min, int max) ;
```

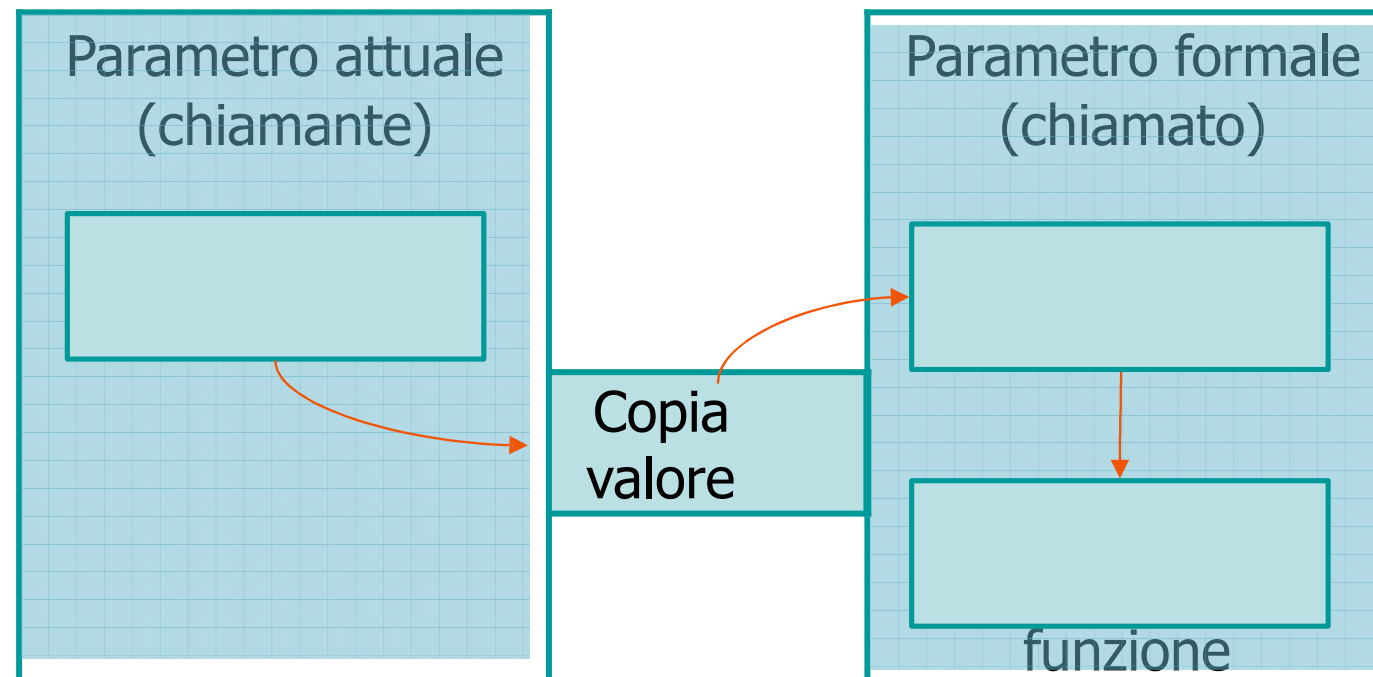
```
int main(void)
{
    int x, a, b ;
    ...
    x = leggi(a, b) ;
    ...
}
```

```
int leggi(int min, int max)
{
    ... codice della funzione ...
}
```

Passaggio dei parametri



- Ogni volta che viene chiamata una funzione, avviene il trasferimento del valore corrente dei parametri attuali ai parametri formali



- La funzione chiamata non ha assolutamente modo di
 - Conoscere il nome delle variabili utilizzate come parametri attuali
 - Ne conosce solo il valore corrente
 - Modificare il valore delle variabili utilizzate come parametri attuali
 - Riceve solamente una copia del valore
- Questo meccanismo è detto passaggio "by value" dei parametri
 - È l'unico possibile in C

Errore frequente



- Immaginare che una funzione possa modificare i valori delle variabili



```
void azzera(int x)
{
    x = 0 ;
}
```

Parametri di tipo vettoriale



- Il meccanismo di passaggio "by value" è chiaro nel caso di parametri di tipo scalare
- Nel caso di parametri di tipo array (vettore o matrice), il linguaggio C prevede che:
 - Un parametro di tipo array viene passato trasferendo una copia dell'indirizzo di memoria in cui si trova l'array specificato dal chiamante
 - Passaggio "by reference"

- Nel passaggio di un vettore ad una funzione, il chiamato utilizzerà l'indirizzo a cui è memorizzato il vettore di partenza
- La funzione potrà quindi modificare il contenuto del vettore del chiamante
- Maggiori dettagli nella prossima lezione

- All'interno del corpo di una funzione è possibile definire delle **variabili locali**

```
int leggi(int min,  
          int max)  
{  
    int v ;  
    scanf("%d", &v) ;  
    return v ;  
}
```

- Le variabili locali sono accessibili solo dall'interno della funzione
- Le variabili locali sono indipendenti da eventuali variabili di ugual nome definite nel main
- In ogni caso, dal corpo della funzione è impossibile accedere alle variabili definite nel main
- Le variabili locali devono avere nomi diversi dai parametri formali

Istruzioni eseguibili



- Il corpo di una funzione può contenere qualsiasi combinazione di istruzioni eseguibili
- Ricordare l'istruzione return

```
int leggi(int min,  
          int max)  
{  
    int v ;  
    scanf("%d", &v) ;  
    return v ;  
}
```

Parametri “by reference”



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Introduzione
- Operatori & e *
- Passaggio “by reference”
- Passaggio di vettori
- Esercizio “strcpy”

Passaggio dei parametri



- Il linguaggio C prevede il passaggio di parametri “by value”
- Il chiamato non può modificare le variabili del chiamante
- Il parametro formale viene inizializzato con una copia del valore del parametro attuale

- Il passaggio "by value" risulta inefficiente qualora le quantità di dati da passare fossero notevoli
 - Nel caso del passaggio di vettori o matrici, il linguaggio C non permette il passaggio "by value", ma copia solamente l'indirizzo di partenza
 - Esempio: `strcmp`
- Talvolta è necessario o utile poter modificare il valore di una variabile nel chiamante
 - Occorre adottare un meccanismo per permettere tale modifica
 - Esempio: `scanf`

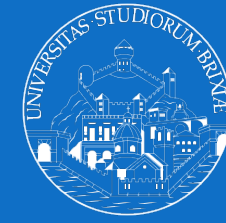
- La soluzione ad entrambi i problemi è la stessa: Nel passaggio di vettori, ciò che viene passato è solamente l'indirizzo
- Per permettere di modificare una variabile, se ne passa l'indirizzo, in modo che il chiamato possa modificare direttamente il suo contenuto in memoria
- Viene detto passaggio "by reference" dei parametri
- Definizione impropria, in quanto gli indirizzi sono, a loro volta, passati "by value"

Operatori sugli indirizzi



- Per gestire il passaggio "by reference" dei parametri occorre
 - Conoscere l'indirizzo di memoria di una variabile
 - Operatore &
 - Accedere al contenuto di una variabile di cui si conosce l'indirizzo ma non il nome
 - Operatore *
- Prime nozioni della aritmetica degli indirizzi, che verrà approfondita in Unità successive

Operatore &



➤ L'operatore **indirizzo-di** restituisce l'indirizzo di memoria della variabile a cui viene applicato

- **&a** è l'indirizzo 1012
- **&b** è l'indirizzo 1020

	1000	
	1004	
	1008	
int a →	1012	37
	1016	
int b →	1020	-4
	1024	
	1028	

- L'indirizzo di una variabile viene deciso dal compilatore
- L'operatore & si può applicare solo a variabili singole, non ad espressioni
 - Non ha senso `&(a+b)`
 - Non ha senso `&(3)`
- Conoscere l'indirizzo di una variabile permette di leggerne o modificarne il valore senza conoscerne il nome

Variabili “puntatore”



- Per memorizzare gli indirizzi di memoria, occorre definire opportune variabili di tipo “indirizzo di...”
- Nel linguaggio C si chiamano **puntatori**
- Un puntatore si definisce con il simbolo *****
 - `int *p ; /* puntatore ad un valore intero */`
 - `float *q ; /* puntatore ad un valore reale */`

Esempio



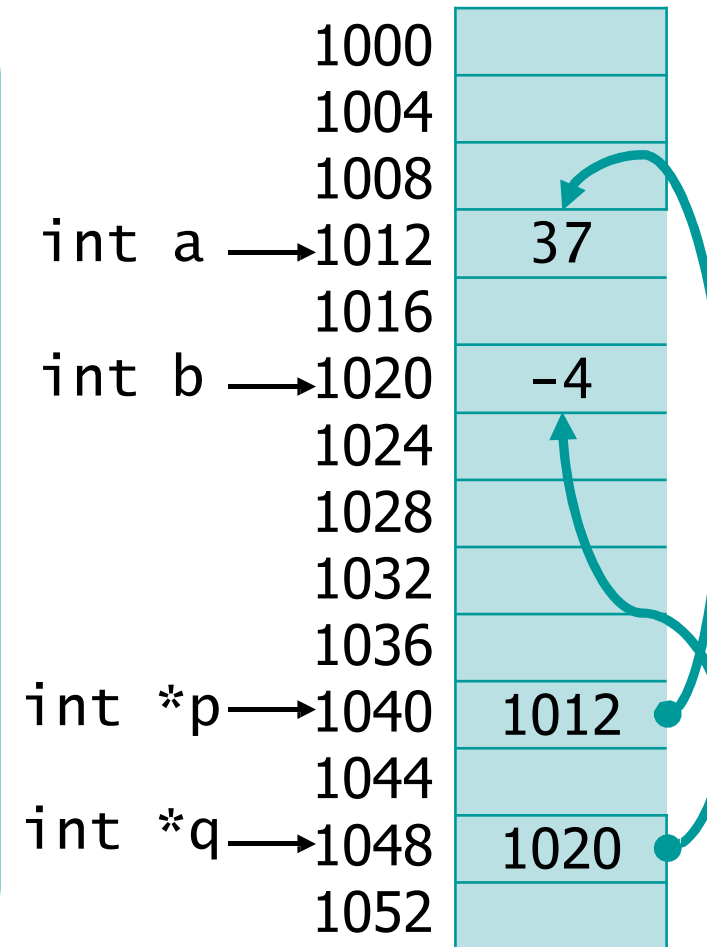
```
int main(void)
{
    int a, b ;
    int *p, *q ;

    a = 37 ;
    b = -4 ;

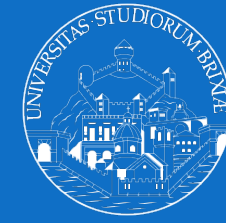
    p = &a ;
    /* p "punta a" a */

    q = &b ;
    /* q "punta a" b */

}
```

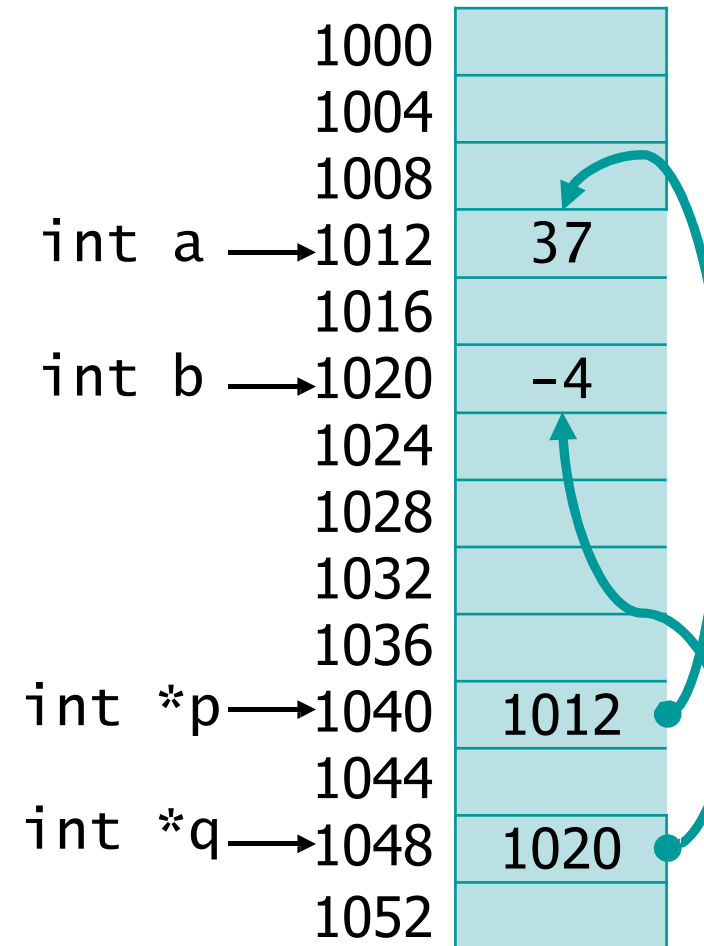


Operatore *

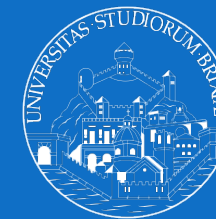


► L'operatore di **accesso indiretto** permette di accedere, in lettura o scrittura, al valore di una variabile di cui si conosce l'indirizzo

- ***p** equivale ad **a**
- ***q** equivale a **b**
- ***p = 0 ;**
- **if(*q > 0) ...**



Costrutti frequenti



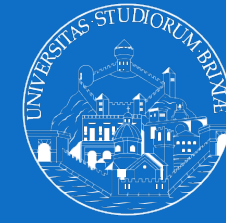
Costrutto	Significato
<code>int x ;</code>	x è una variabile intera
<code>int *p ;</code>	p è un puntatore a variabili intere
<code>p = &x ;</code>	p punta ad x
<code>*p = 0 ;</code>	Azzera la variabile puntata da p (cioè x)
<code>b = *p ;</code>	Leggi il contenuto della variabile puntata da p e copialo in b

Passaggio “by reference”



- Obiettivo: passare ad una funzione una variabile, in modo tale che la funzione la possa modificare
- Soluzione:
 - Definire un parametro attuale di tipo puntatore
 - Al momento della chiamata, passare l'indirizzo della variabile (anziché il suo valore)
 - All'interno del corpo della funzione, fare sempre accesso indiretto alla variabile di cui è noto l'indirizzo

Esempio: “Azzera”



- Scrivere una funzione `azzera`, che riceve un parametro di tipo intero (by reference) e che azzera il valore di tale parametro

Soluzione



```
void azzera( int *v ) ;
```

```
int main( void )  
{  
    int x ;  
    ...  
    azzera(&x) ;  
    ...  
}
```

```
void azzera( int *v )  
{  
    *v = 0 ;  
}
```

Esempio: “Scambia”



- Scrivere una funzione `scambia`, che riceve due parametri di tipo intero (by reference) e che scambia tra di loro i valori in essi contenuti

Soluzione



```
void scambia( int *p, int *q ) ;
```

```
int main( void )  
{  
    int a,b ;  
    ...  
    scambia(&a, &b) ;  
    ...  
}
```

```
void scambia( int *p, int *q )  
{  
    int t ;  
    t = *p ;  
    *p = *q ;  
    *q = t ;  
}
```

- Il meccanismo di passaggio by reference spiega (finalmente!) il motivo per cui nella funzione `scanf` è necessario specificare il carattere & nelle variabili lette
- Le variabili vengono passate by reference alla funzione `scanf`, in modo che questa possa scrivervi dentro il valore immesso dall'utente

Passaggio di vettori e matrici

- Nel linguaggio C, il nome di un array (vettore o matrice) è automaticamente sinonimo del puntatore al suo primo elemento

```
int main(void)
{
    int v[10] ;
    int *p ;

    p = & v[0] ;

}
```



p e v sono
del tutto
equivalenti

- Quando il parametro di una funzione è di tipo array (vettore o matrice)
 - L'array viene passato direttamente "by reference"
 - Non è necessario l'operatore & per determinare l'indirizzo
 - È sufficiente il nome del vettore
 - Non è necessario l'operatore * per accedere al contenuto
 - È sufficiente l'operatore di indicizzazione []
 - Non è possibile, neppure volendolo, passare un array "by value"

Esercizio “Duplicati”



- Scrivere una funzione che, ricevendo due parametri
 - Un vettore di `double`
 - Un intero che indica l'occupazione effettiva di tale vettore

possa determinare se vi siano valori duplicati in tale vettore

- La funzione ritornerà un intero pari a 1 nel caso in cui vi siano duplicati, pari a 0 nel caso in cui non ve ne siano

Soluzione (1/3)



```
int duplicati(double v[], int N) ;  
/*  
Riceve in ingresso il vettore v[] di double  
che contiene N elementi (da v[0] a v[N-1])  
  
Restituisce 1 se in v[] non vi sono duplicati  
Restituisce 2 se in v[] vi sono duplicati  
  
Il vettore v[] non viene modificato  
*/
```

Soluzione (2/3)



```
int duplicati(double v[], int N)
{
    int i, j ;

    for(i=0; i<N; i++)
    {
        for(j=i+1; j<N; j++)
        {
            if(v[i]==v[j])
                return 1 ;
        }
    }
    return 0 ;
}
```

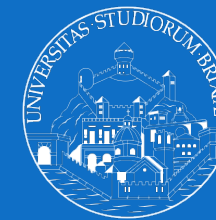
Soluzione (3/3)



```
int main(void)
{
    const int MAX = 100 ;
    double dati[MAX] ;
    int Ndati ;
    int dupl ;

    ...
    dupl = duplicati(dati, Ndati) ;
    ...
}
```

Errore frequente



- Nel passaggio di un vettore occorre indicarne solo il nome



```
dupl = duplicati(dati, Ndati) ;
```

```
dupl = duplicati(dati[], Ndati) ;
```

```
dupl = duplicati(dati[MAX], Ndati) ;
```

```
dupl = duplicati(dati[Ndati], Ndati) ;
```

```
dupl = duplicati(&dati, Ndati) ;
```

- Nel caso dei vettori, il linguaggio C permette solamente il passaggio by reference
- Ciò significa che il chiamato ha la possibilità di modificare il contenuto del vettore
- Non è detto che il chiamato effettivamente ne modifichi il contenuto
- La funzione duplicati analizza il vettore senza modificarlo
- Esplicitarlo sempre nei commenti di documentazione

Esercizio “strcpy”



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Si implementi, sotto forma di funzione, la ben nota funzione di libreria `strcpy` per la copia di due stringhe

Soluzione (1/2)



```
void strcpy(char *dst, char *src) ;  
/*
```

Copia il contenuto della stringa src
nella stringa dst

Assume che src sia 0-terminata, e restituisce
dst in forma 0-terminata.

Assume che nella stringa dst vi sia spazio
sufficiente per la copia.

La stringa src non viene modificata.
*/

Soluzione (2/2)



```
void strcpy(char *dst, char *src)
{
    int i ;

    for(i=0; src[i]!=0; i++)
    {
        dst[i] = src[i] ;
    }
    dst[i] = 0 ;
}
```


- La funzione può essere dichiarata in due modi:
 - `void strcpy(char *dst, char *src)`
 - `void strcpy(char dst[], char src[])`
- Sono forme assolutamente equivalenti
- Tutte le funzioni di libreria che lavorano sulle stringhe accettano dei parametri di tipo `char *`