

Unità Didattica 4

Linguaggio C

Vettori. Puntatori.
Funzioni: passaggio di parametri
per indirizzo.

Vettori

- Struttura astratta:
 - Insieme di elementi dello stesso tipo, ciascuno individuato da un indice;
- Struttura concreta in C:
 - Struttura di elementi dello stesso nome e tipo, allocati in locazioni di memoria contigue;
 - Nella loro definizione statica hanno la medesima dimensione per tutta l'esecuzione del programma;
 - Per fare riferimento ad un elemento si specifica:
 - Nome del vettore;
 - Numero di posizione all'interno del vettore stesso;

Dichiarazione e accesso ad un vettore

- Dichiarazione: `tipo nomeVettore[dimensione];`
Es: `int a[10];`
 - La dichiarazione di variabili vettoriali segue le stesse regole di scope e di ciclo di vita delle variabili scalari;
 - Gli elementi sono inizializzati a 0 se il vettore è globale, non sono inizializzati se è locale;
- Dichiarazione e inizializzazione contestuali:
Es: `int b[] = {7, 3, -2};`
- Accesso:
 - Dato un vettore v di n elementi, il primo elemento di v ha indice 0, gli altri elementi hanno indici interi crescenti:
`v[0], v[1], ..., v[n-1]`

Esempio di uso di vettori: Somma elementi di un vettore

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

void main(void)
{
    float v[]={3.2,4.3,-7.8,5,-3.14};
    int i;
    float somma = 0;

    for (i = 0; i < 5; i++)
        somma = somma + v[i];

    printf("somma %f\n", somma);
}
```

Vettori multidimensionali

- Struttura dati di elementi individuati da più indici (per es. un vettore bidimensionale: struttura dati a 2 indici, matrice);

- Dichiarazione:

```
float a[2][3];
```

- Esempio di inizializzazioni:

```
float a[][] = {{-1.2, 3, 0}, {7, -2.3, 8}};
```

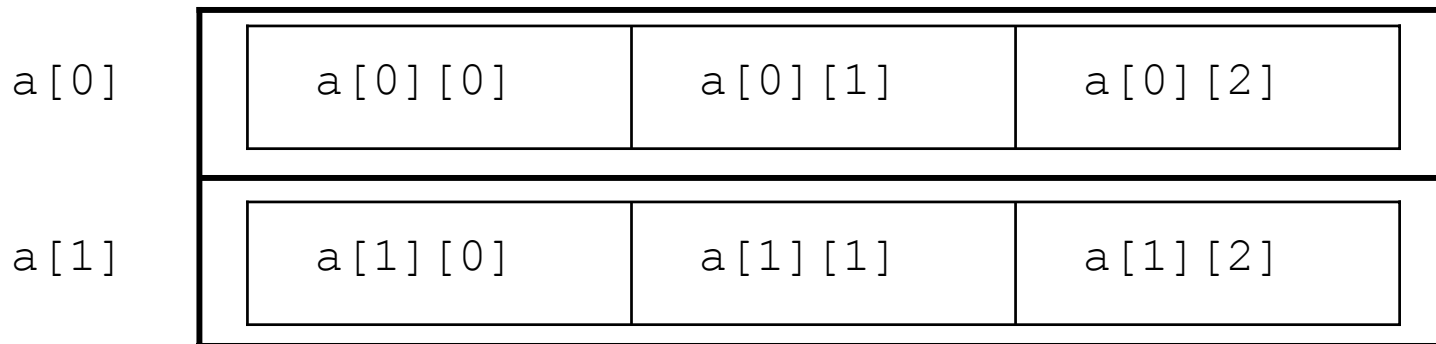
oppure:

```
for (i = 0; i < 2; i++)  
    for (j = 0; j < 3; j++)  
        scanf("%f", &(a[i][j]));
```

Vettori bidimensionali

- In C un vettore bidimensionale è un vettore ad una dimensione in cui ogni elemento è un vettore;

```
float a[2][3];
```



- Accesso agli elementi (visualizzazione per righe):

```
for (i = 0; i < 2; i++)  
{  
    for(j = 0; j < 3; j++)  
        printf("%2.3f ", a[i][j]);  
    printf("\n");  
}
```

Puntatori

- Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile;
- A differenza delle variabili *normali*, che contengono valori di un insieme numerico, le variabili puntatore contengono indirizzi (riferimenti) a variabili contenenti dati;
- Sono utilizzati per:
 - Passaggio di parametri per indirizzo alle funzioni;
 - Riferimento a strutture dati.

Dichiarazione di puntatori

- Viene utilizzato il carattere `*`
 - Es: `int *p;`
dichiara una variabile puntatore, di nome `p`, ad una variabile di tipo `int` (si legge da destra a sinistra);
- Dichiarazioni multiple: è sufficiente usare un `*` prima della dichiarazione di ogni variabile;
 - Es: `int *a, *b;`
- E' possibile dichiarare puntatori a qualsiasi tipo di dato;
- E' possibile inizializzare i puntatori ad un indirizzo oppure a `NULL`.

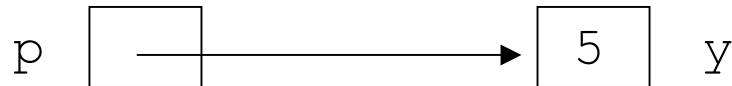
Operatori unari “&” e “*”

- & (operatore indirizzo): applicato ad una variabile, restituisce l'indirizzo della variabile

Es: `int y = 5;`

`int *p;`

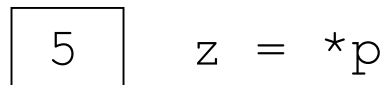
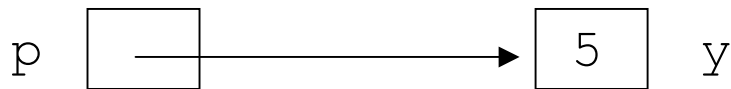
`p = &y; /*assegna a p l'indirizzo di y, "p punta ad y"*/`



- * (operatore indirezione): applicato ad un puntatore, restituisce il contenuto della variabile da esso puntata

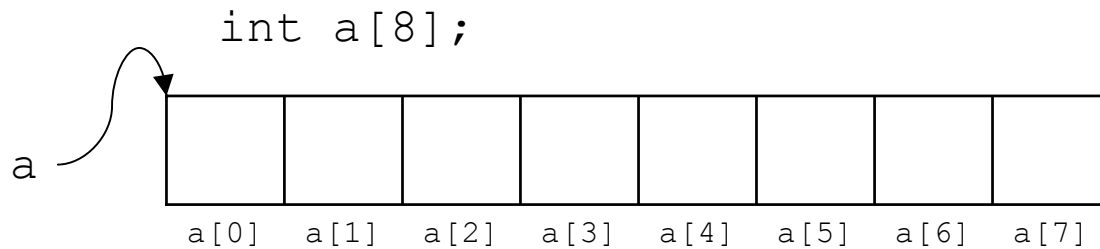
Es: `int z;`

`z = *p;`

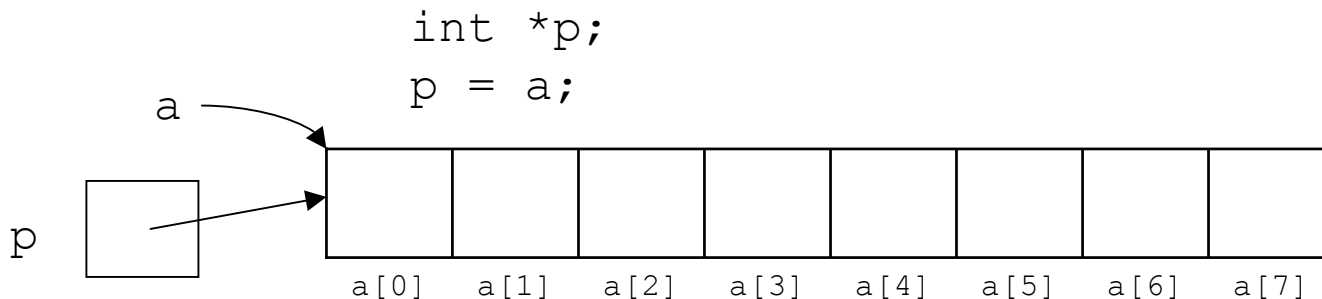


Vettori e Puntatori

- Il nome di un vettore è una costante e rappresenta l'indirizzo della prima cella di memoria del vettore stesso;



- Tale indirizzo può essere assegnato ad una variabile puntatore, poiché essa può contenere un indirizzo;



Aritmetica dei Puntatori

```
int a[10], *p, x, y;

x = 1;
p = &x;          //p punta a x
y = *p;         //y vale 1

*p = 0;         //x viene posto a 0

p = &a[0];      //p punta ad a[0]
p = &a[1];      //p punta ad a[1]

*p = *p + 10;   //si somma 10 al contenuto di ciò che è
                //puntato da p
(*p)++;        //Corrisponde a (++*p) incremento di 1
                //del contenuto di ciò che è puntato da p

p = a;          //p punta ad a[0]

*(p+5);        //Corrisponde a a[5] se p punta ad a
```

Aritmetica dei Puntatori

```
int b[2][3];  
int *p;  
  
p = b+1;           //In p c'e' l'indirizzo della riga di indice 1  
  
*(p+2);           //Visualiz contenuto elem b[1][2]
```

Passaggio di parametri alle funzioni: Passaggio per indirizzo

- In C i parametri sono passati solo per valore;
- Se il valore passato è un indirizzo si ottiene il passaggio di parametri per indirizzo;
- Il passaggio per indirizzo consente alla funzione di operare sulla stessa locazione di memoria del parametro che le viene passato al momento della chiamata;
- Tale modalità consente:
 - di alterare il valore di una variabile anche nella funzione chiamante;
 - di avere un altro modo per restituire il risultato di una funzione.

Esempio di passaggio parametri: Passaggio per valore

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

```
void main(void)
```

```
{
```

```
    int a;
```

```
    a = 5;
```

```
    esempio_funz(a);
```

```
    printf("%d\n", a);
```

```
}
```

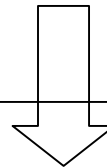
```
void esempio_funz(int b)
```

```
{
```

```
    b = b + 1;
```

```
}
```

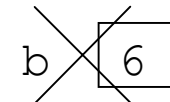
a 5 Quando il controllo ritorna al
main a == 5



b 5

Al termine della funzione
la variabile b scompare

b = b + 1



Esempi di passaggio parametri: Passaggio per indirizzo

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

```
void main(void)
```

```
{
```

```
    int a;
```

```
    a = 5;
```

```
    esempio_funz(&a);
```

```
    printf("%d\n", a);
```

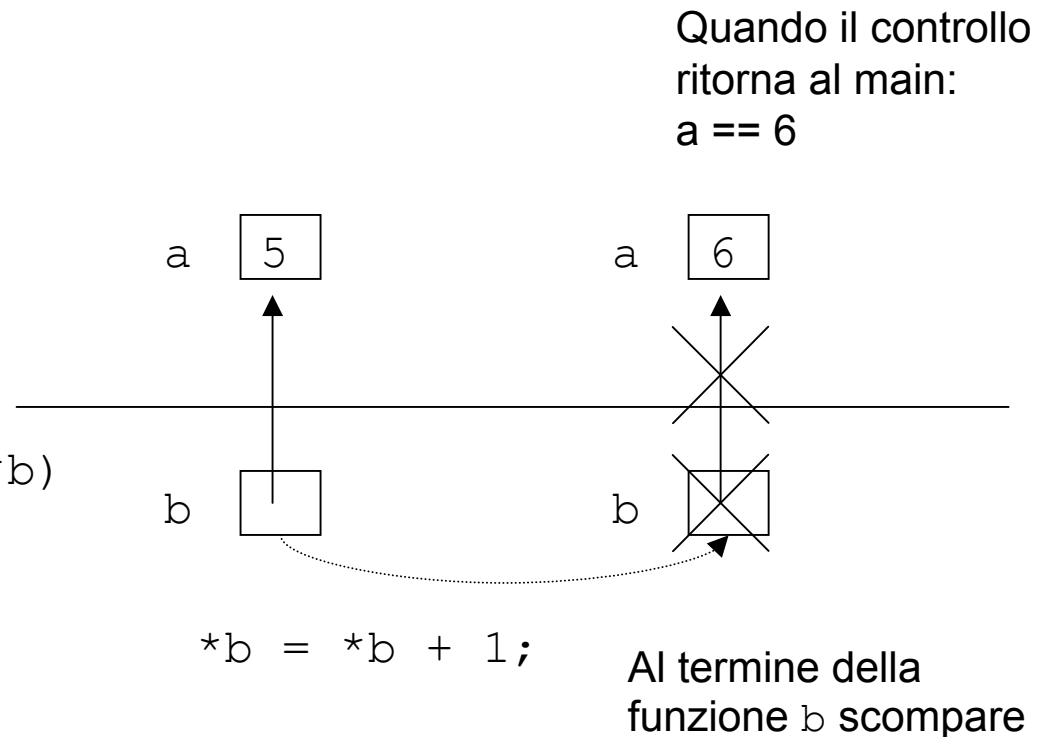
```
}
```

```
void esempio_funz(int *b)
```

```
{
```

```
    *b = *b + 1;
```

```
}
```



Esempio di passaggio parametri

```
void main(void)
{
    int a=3, b=7;
    scambia1(a,b);    //Al termine di scambia1  a 3    b 7

    scambia2(&a,&b); //Al termine di scambia2  a 7    b 3
}
void scambia1(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
void scambia2(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Allocazioni statiche e dinamiche

- Allocare una struttura dati significa riservarle spazio (byte) in memoria centrale;
- Con l'allocazione statica, il compilatore, al momento della dichiarazione e per tutto il loro ciclo di vita, riserva uno spazio in byte sufficiente a contenere le variabili allocate (scalari o vettoriali) di un determinato tipo;

```
Es: int a; //si riservano byte necessari per un int
    float v[10]; //byte necessari per 10 float
```
- Con l'allocazione dinamica è possibile:
 - Decidere il momento in cui allocare e deallocare la struttura dati;
 - Decidere la quantità di memoria strettamente necessaria da riservare alla struttura dati;

Allocazioni dinamiche

- Si possono ottenere mediante la funzione di libreria contenuta in `<stdlib.h>`

```
void *malloc(n° byte della struttura dati da  
allocare);
```

- `malloc` restituisce l'indirizzo di base della struttura dati allocata;
- Per ottenere la dimensione in byte della struttura dati da allocare esiste la funzione:

```
n° byte = sizeof(tipo di dato);
```

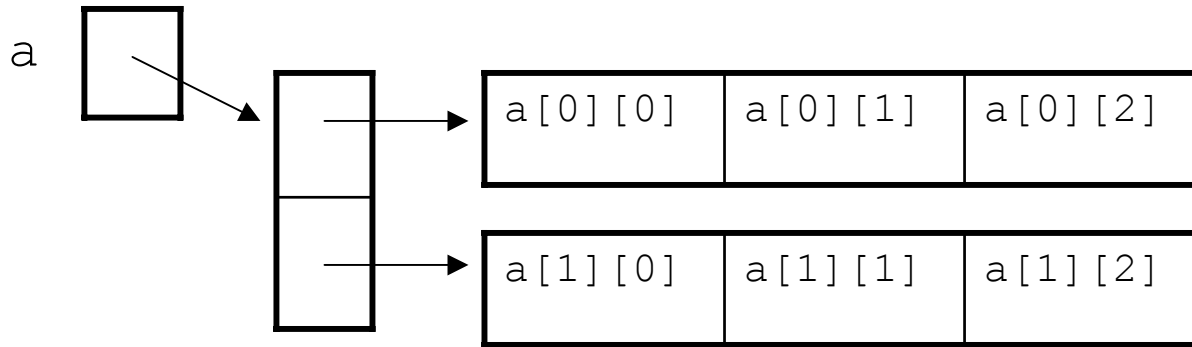
- Per liberare la memoria allocata si usa la funzione:

```
free(puntatore alla struttura dati);
```

Esempio di allocazione e deallocazione dinamica di un vettore

```
//Alloc. dinamica di un vettore di 10 interi  
int *v;  
v = (int *)malloc(10 * sizeof(int));  
...  
  
//Deallocazione della memoria associata a v  
free(v);
```

Allocazione dinamica di un array bidimensionale 2x3



```
int **a; // "a" è un puntatore ad un puntatore a int
int i;
a = (int **)malloc(sizeof(int *) * 2);
for (i = 0; i < 2; i++)
{
    a[i] = (int *)malloc(sizeof(int) * 3);
}
```

Passaggio di un vettore ad una funzione

- Si può ottenere passando alla funzione l'indirizzo della prima locazione di memoria;
- Es:

```
void main(void)
{ int a[10];          //Allocazione statica
  float *v;
  ...
  v = (float *)malloc(sizeof(float) * 20); //Allocaz.
                                           //dinamica
  funz(a, v);
  ...
}
void funz(int *b, float *w)
{
    //Usando "b" e "w" si accede rispettivamente alle aree
    //di memoria di "a" e "v";
}
```

Passaggio di una matrice ad una funzione

```
void main(void)
{ int a[2][3];
  ...
  funz(a);
  ...
}
```

```
void funz(int **b)
{
  ...
  printf("%d\n", b[1][2]); //Visual. contenuto elem. b[1][2]
}

```