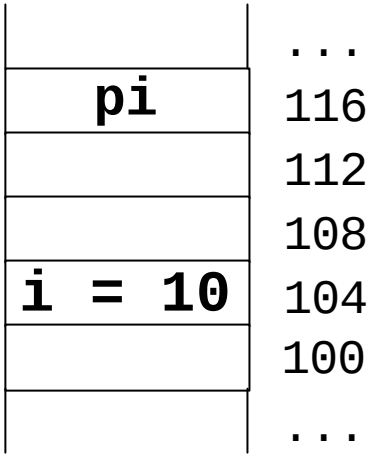
Puntatori

```
int i = 10;

int * pi = &i;
```



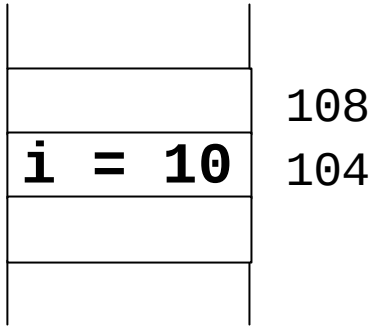
`int * pi` = **pi** contiene un'informazione che mi permette di accedere ("puntare") ad una **variabile intera**

Un puntatore contiene un numero che indica la **locazione di memoria** dove è presente la **variabile** puntata

pi contiene un numero che indica la **locazione di memoria** dove è presente la **variabile** intera **i** puntata

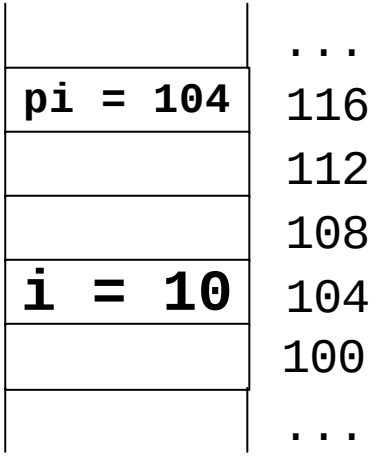
Puntatori

```
int i = 10;
int * pi = &i;
```



"&" = operatore "indirizzo di …"

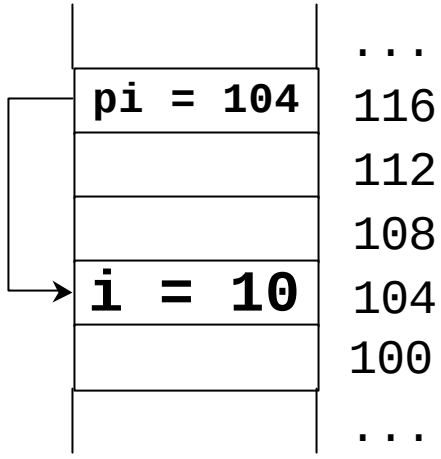
&i = 104 → pi = 104



Un puntatore *contiene un numero che indica la* ***locazione di memoria*** *dove è presente la* ***variabile*** *puntata*

Accesso tramite puntatori

```
int i = 10;
int * pi = &i;
```



La scrittura:

***pi**

posta all'interno di un codice C, significa "accesso all'informazione contenuta nella variabile puntata da pi"

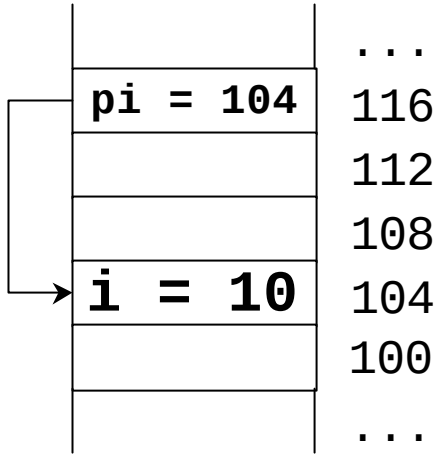
`pi = 104` → "punta" alla locazione 104 →
→la loc. 104 contiene la variabile "i" il cui valore è 10 →
→ ***pi = 10**

Un puntatore *contiene un numero che indica la* ***locazione di memoria*** *dove è presente la* ***variabile*** *puntata*

Accesso tramite puntatori

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i = 10;
  int * pi = &i;
  printf (“Il valore di ‘i’ è %d\n”, i);
  printf (“... tramite ‘pi’ è %d\n”, *pi);
  *pi = *pi + 20; // incremento di 20
  printf (“Il nuovo valore di ‘i’ è %d\n”, i);
  printf (“... tramite ‘pi’ è %d\n”, *pi);
}
```



Il valore di ‘i’ è 10
... tramite ‘pi’ è 10
Il nuovo valore di ‘i’ è 30
... tramite ‘pi’ è 30

Tipo (**pi**) = **int ***
Tipo (***pi**) = **int**

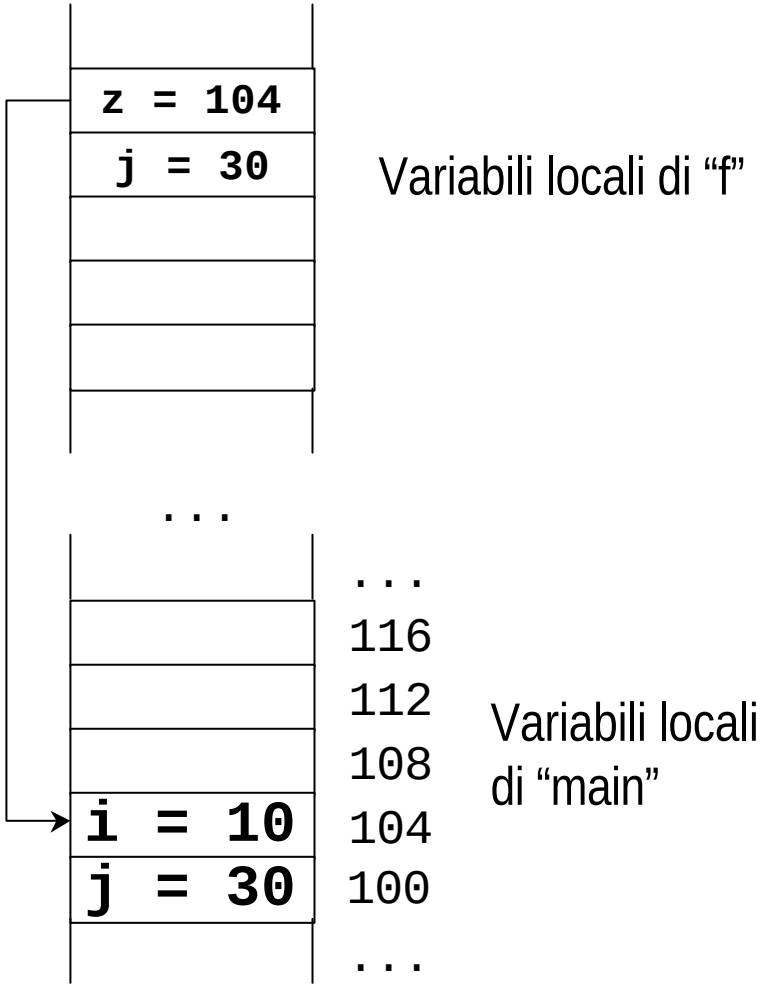
Passaggio per riferimento

```
#include <stdio.h>

void f (int * z, int j)
{
  *z = *z + j;
  j *= 2;
}

int main(int argc, char *argv[])
{
  int i = 10;
  int j = 30;
  printf (“i = %d, j = %d\n”, i, j);
  f (&i, j);
  printf (“i = %d, j = %d\n”, i, j);
}
```

i = 10, j = 30
i = 40, j = 30

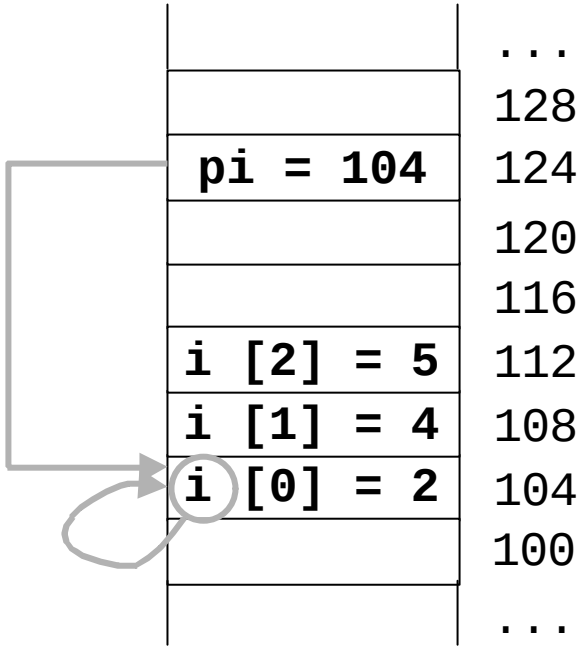


Vettori

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i[3];
  int *pi;
  i[0] = 2;
  i[1] = 4;
  i[2] = 5;
  printf (“i[0] = %d\n”, i[0]);
  printf (“*i = %d\n”, *i);
  pi = i;
  printf (“*pi = %d\n”, pi[0]);
}
```

i[0] = 2
***i = 2**
pi[0] = 2



Tipo (**i[x]**) = **int**
Tipo (**i**) = **int ***

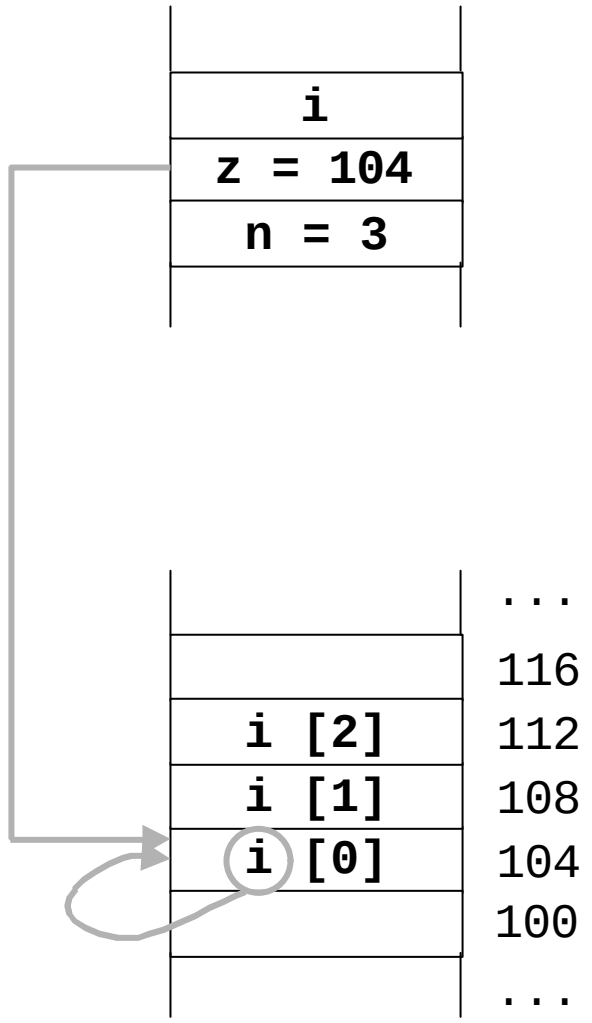
Vettori e passaggio per riferimento

```
#include <stdio.h>

void f(int * z, int n)
{
  int i;
  for (i = 0;i < n;i++)
    z[i] = i * 2;
}

int main(int argc, char *argv[])
{
  int i[3], k;
  f (i, 3);
  for (k = 0; k < 3; k++)
    printf (“i[%d] = %d\n”, k, i[k]);
}
```

i[0] = 0
i[1] = 2
i[2] = 4

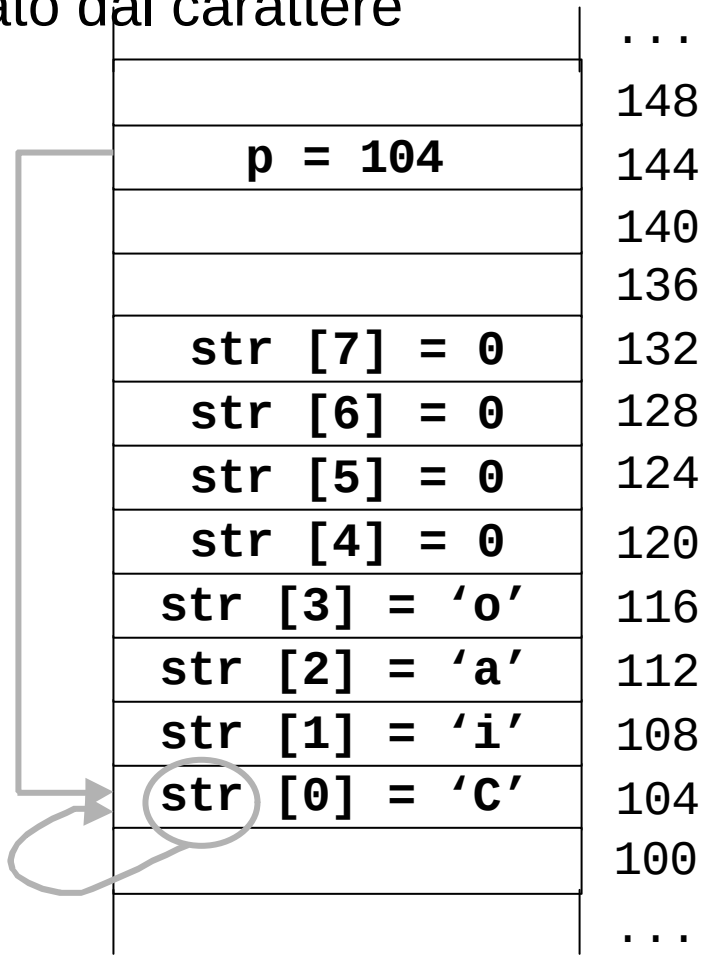


Stringhe

Una stringa in C è un ***array di caratteri*** terminato dal carattere ***ASCII 0*** (NUL)

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

int main(int argc, char *argv[])
{
  char str[8];
  char * p;
  strcpy (str, “Ciao”);
  printf (“str = %s\n”, str);
  p = str;
  printf (“p = %s, *p = %c\n”, p, *p);
}
```



str = ciao
p = ciao, *p = c

Strcpy = “string copy”

```
int strcpy (char * dest, char * src);
```

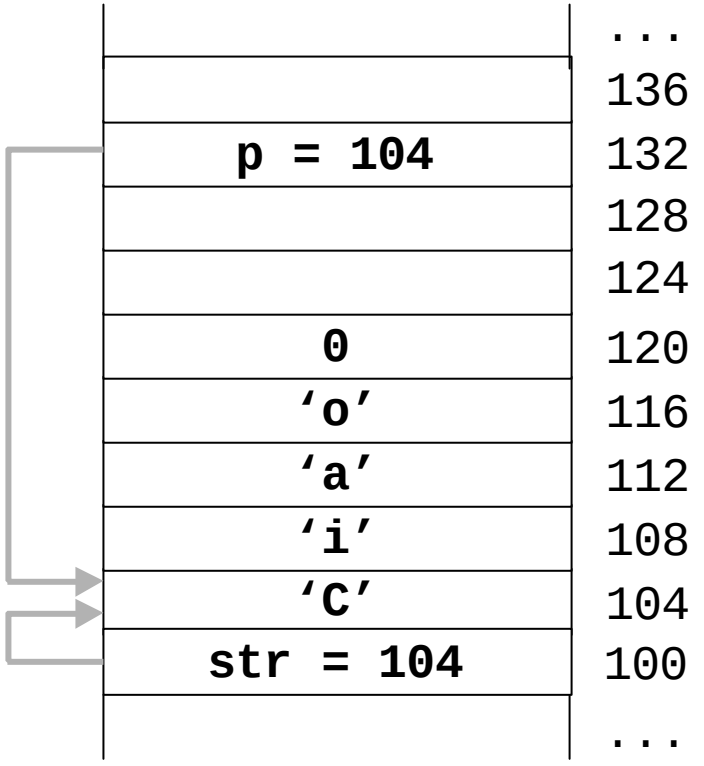
Stringhe

Una stringa in C è un ***array di caratteri*** terminato dal carattere ***ASCII "0"***

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  char * str = "Ciao";
  char * p;
  printf ("str = %s\n", str);
  p = str;
  printf ("p = %s, *p = %c\n", p, *p);
}
```

str = ciao
p = ciao, *p = c

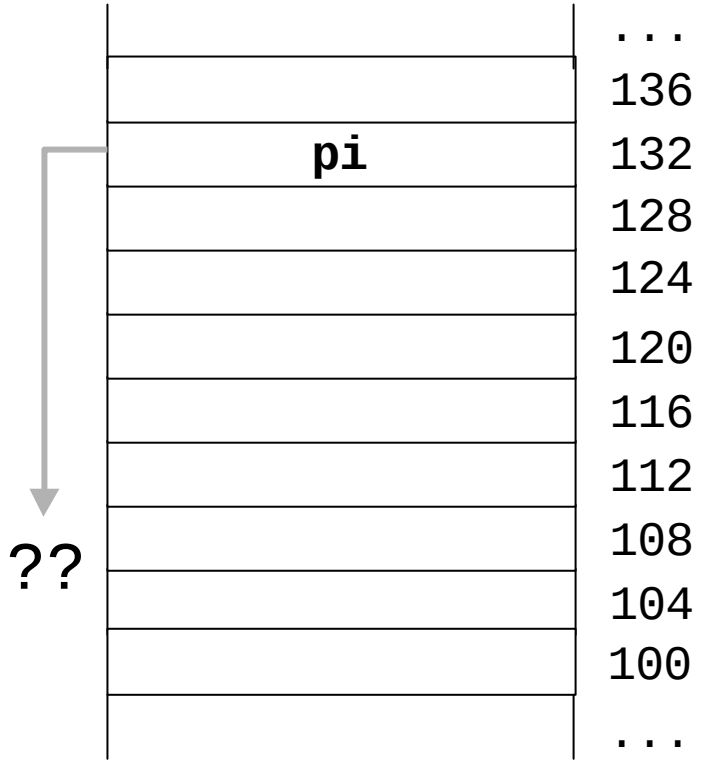


Puntatori non allocati/non inizializzati

```
int scanf (char * fmt, <<puntatori alle variabili>>)
```

Inserimento di un intero:

```
int main(int argc, char * argv[])
{
  int * pi;
  scanf("%d", pi);
  printf ("%d\n", *pi);
}
```

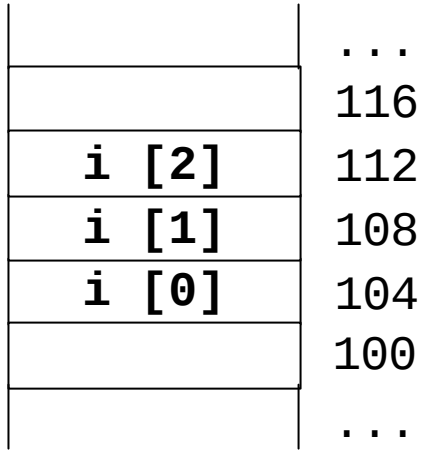


Il programma va in "crash" perché la scanf tenta di riferire la zona puntata da **pi**, ma quest'ultimo ***non è inizializzato!***

Allocazione dinamica

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i[3];
…
}
```



La dimensione del vettore è specificata nel sorgente ed è quindi nota a “compile time”

→ Il vettore è allocato in memoria ***staticamente***

A volte non è possibile stabilire a priori le dimensioni di un vettore, le quali saranno note a “run time” (durante l’esecuzione del programma). E’ quindi necessario disporre di un meccanismo che permetta di ***creare dinamicamente*** un vettore delle dimensioni volute.

Allocazione dinamica

Per l'allocazione dinamica si usa la funzione:

```
void * malloc (int size);
```

che permette di "allocare" (riservare) una zona di memoria, delle dimensioni specificate, restituendone l'indirizzo (puntatore)

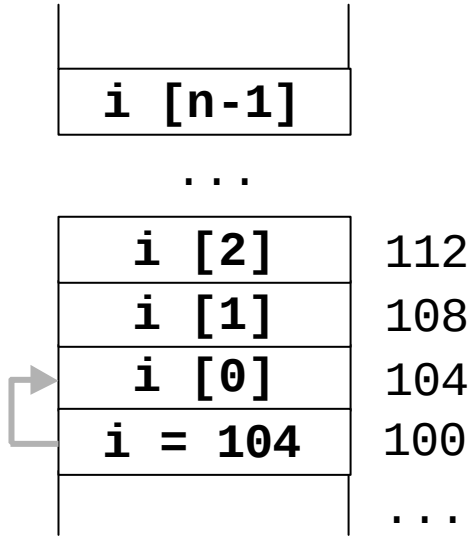
Allocazione di un vettore di interi di dimensione "n"

```
int * i;
i = (int *) malloc (n * sizeof (int) );
```

"Conversione di tipo" da "void *" *a* "int *"

Numero di elementi

Dimensione di ogni elemento



Esempio di allocazione dinamica: ordinamento

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

int main(int argc, char * argv[])
{
  int * array;
  int n, i;
  printf (“Inserisci il numero di elementi “);
  scanf (“%d”, &n);
  array = (int *) malloc (n * sizeof (int));
  if (array == NULL) {
    printf (“Impossibile allocare la memoria richiesta\n”);
    exit (1);
  }
  for (i = 0;i < n;i++) {
    printf (“Inserisci l’elemento %d ”, i);
    scanf (“%d”, &array [i]);
  }
  bsort (array, n);
  for (i = 0;i < n;i++)
    printf (“elemento %d = %d\n”, i, array [i]);
}
```

Esempio di allocazione dinamica: ordinamento con strutture

```
void bsort (struct studente * x, int n)
{
  int i,j;
  int swap;
  for (i = 0;i < n;i++) {
    for (j = 0;j < n-1;j++) {
      if (x[j] > x[j+1]) {
        swap = x[j];  x[j] = x[j+1]; x[j+1] = swap;
      }
    }
  }
}
```

Puntatori e strutture

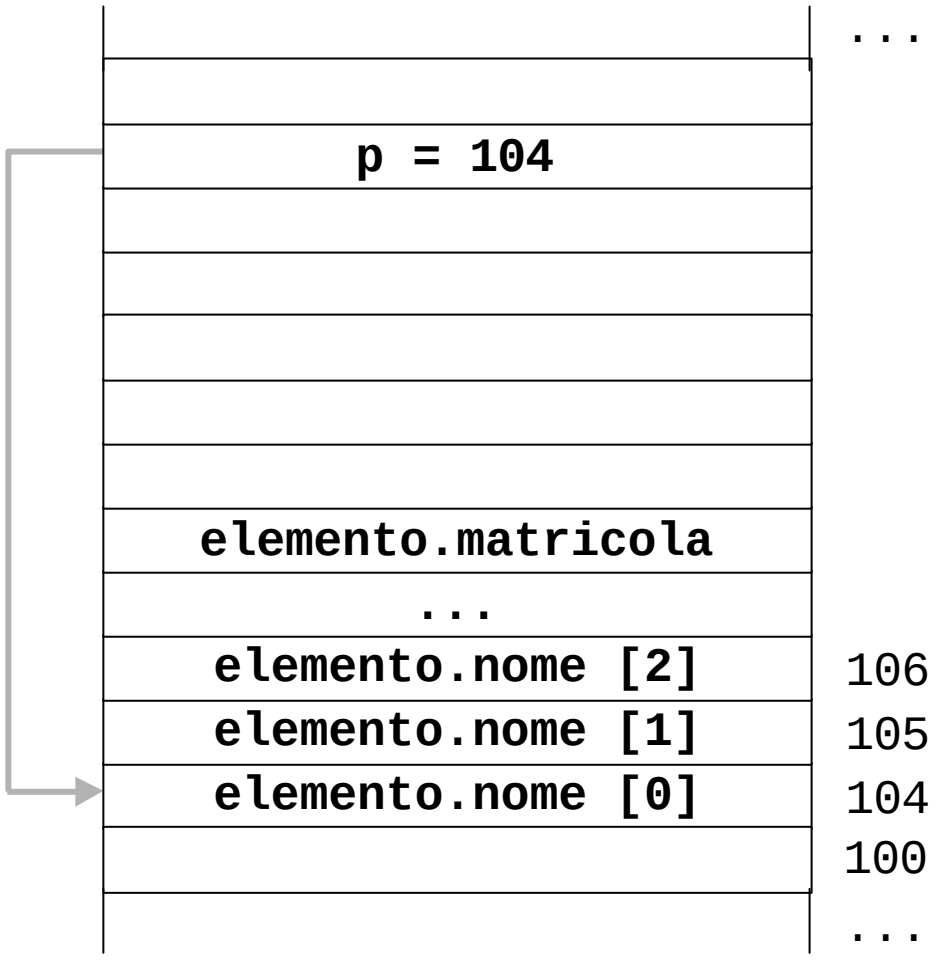
```
struct elem {
  char nome[30];
  int matricola;
} elemento;

struct elem * p;
…

p = &elemento
```

```
Tipo (p) = struct elem *
Tipo (*p) = struct elem
```

```
(*p).nome  (*p).matricola
p->nome    p->matricola
```



Esempio di allocazione dinamica: ordinamento con strutture

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

struct studente {
  char nome [30];
  int matricola;
};

int main(int argc, char * argv[])
{
  struct studente * array;
  int n, i;
  printf (“Inserisci il numero di studenti “);
  scanf (“%d”, &n);
  array = (struct studente *) malloc (n * sizeof (struct studente));
  if (array == NULL) {
    printf (“Impossibile allocare la memoria richiesta\n”);
    exit (1);
  }
…
```

Esempio di allocazione dinamica: ordinamento con strutture

```
…
 for (i = 0;i < n;i++) {
    printf (“Inserisci l’elemento %d\n”, i);
    printf (“Nome “); scanf (“%s”, array [i].nome);
    printf (“Matricola “); scanf (“%d”, &array [i].matricola);
  }
  bsort (array, n);
  for (i = 0;i < n;i++)
    printf (“elemento %d = %s, %d\n”,
             i, array [i].nome, array [i].matricola);
}

void bsort (struct studente * x, int n)
{
  int i,j;
  struct studente swap;
  for (i = 0;i < n;i++) {
    for (j = 0;j < n-1;j++) {
      if (x[j].matricola > x[j+1].matricola) {
        swap = x[j];  x[j] = x[j+1]; x[j+1] = swap;
      }
    }
  }
}
```

Matrici Bidimensionali

```
int m[3][10];     // 3 righe x 10 colonne
```

La memoria e' (ovviamente) **unidimensionale**

→ come e' memorizzata una matrice bidimensionale in memoria?

⇩

Attraverso un **vettore di vettori**

```
int m [3] [10];
```

Ogni elemento e' un ***intero***

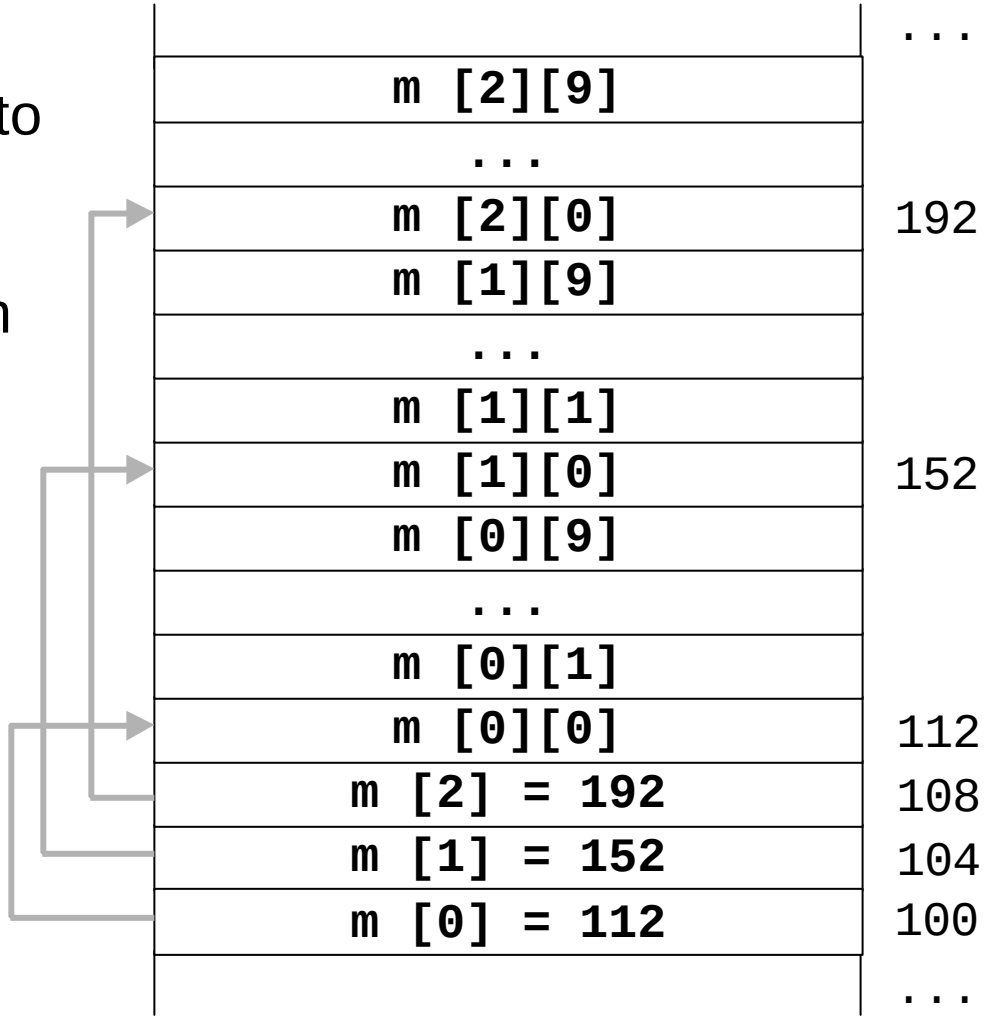
Ogni elemento e' un ***vettore di 10 interi***

Matrici Bidimensionali

```
int m [3] [10];
```

Ogni elemento e' un ***intero***

Ogni elemento e' un ***vettore di 10 interi***




```
Tipo (m[x][y]) = int
Tipo (m[x]) = int *
Tipo (m) = int **
```

Allocazione di Matrici Bidimensionali

```
int m [3] [10];
```

```
Tipo (m[x][y]) = int
Tipo (m[x]) = int *
Tipo (m) = int **
```

Numero di righe noto a compile time (es. 3), num. di colonne noto a runtime.

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

int main(int argc, char * argv[])
{
  int * matrice[3];
  int n, i;
  printf (“Inserisci il numero di colonne “);
  scanf (“%d”, &n);
  for (i = 0; i < 3;i++)
    matrice[i] = (int *)malloc (n * sizeof(int));
}
```

Allocazione di Matrici Bidimensionali

int m [3] [10];

```
Tipo (m[x][y]) = int
Tipo (m[x]) = int *
Tipo (m) = int **
```

Numero di righe e di colonne note a runtime.

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

int main(int argc, char * argv[])
{
  int ** matrice;
  int r, c, i;
  printf ("Inserisci il numero di righe ");
  scanf ("%d", &r);
  printf ("Inserisci il numero di colonne ");
  scanf ("%d", &c);
  matrice = (int **) malloc (r * sizeof (int *));
  for (i = 0; i < r;i++)
    matrice[i] = (int *)malloc (c * sizeof(int));
}
```

Esercizio: Trasposta di una matrice

$$A^T[j,i] = A[j,i]$$

```
int ** trasposta (int **src, int s_rows, int s_cols)
{
  int ** dest;
  int d_cols, d_rows, i, j;
  d_rows = s_cols;
  d_cols = s_rows;
  dest = (int **) malloc (d_rows * sizeof (int *));
  for (i = 0; i < d_rows;i++)
    dest [i] = (int *)malloc (d_cols * sizeof(int));
  for (i = 0; i < s_rows;i++)
    for (j = 0;j < s_cols;j++)
      dest [j][i] = src [i][j];
  return dest;
}
```