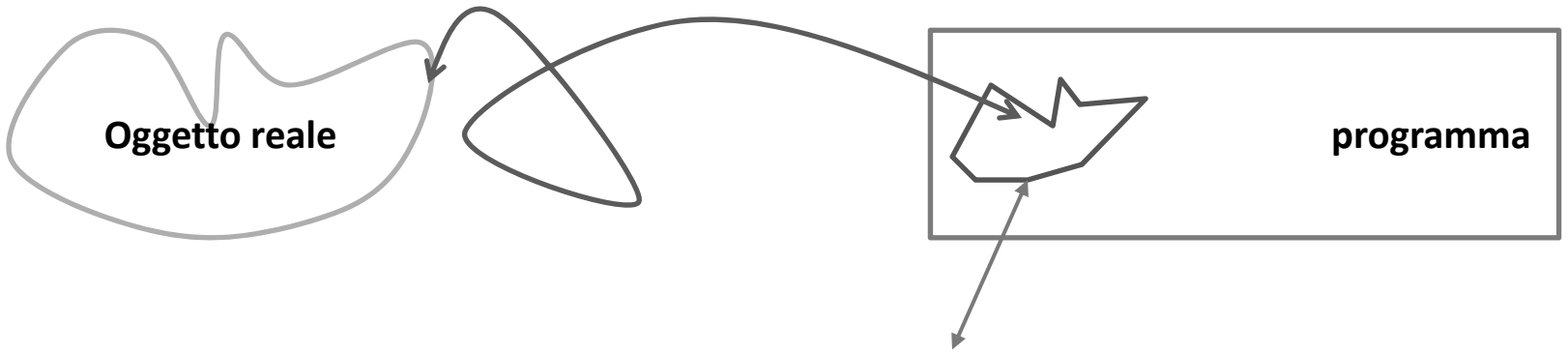
Strutture Dati

Risposta al problema di

- **Rappresentare gli oggetti del problema (mondo reale) nel programma**



Oggetto nel programma = VARIABILE DI UN TIPO
(o insieme di var di uno o piu` tipi)

TIPI e COSTRUTTI di TIPO

- **TIPI BASE**
  - **int, char, float, double** … abbiamo visto la rappresentazione dei valori di questi tipi, e le funzionalità disponibili (gli operatori su valori di questi tipi)

- **TIPI ADDIZIONALI**
  - **definiti dal programmatore, mediante *"costruttori di tipo"*, come [], *, …**
    - **int *, double * … int[] … char[]**
    - **abbiamo visto i principi in base ai quali i "valori" di questi tipi sono rappresentati**
    - **funzionalità (operatori per accedere ai dati: [], *p, &v … aritmetica dei puntatori …)**

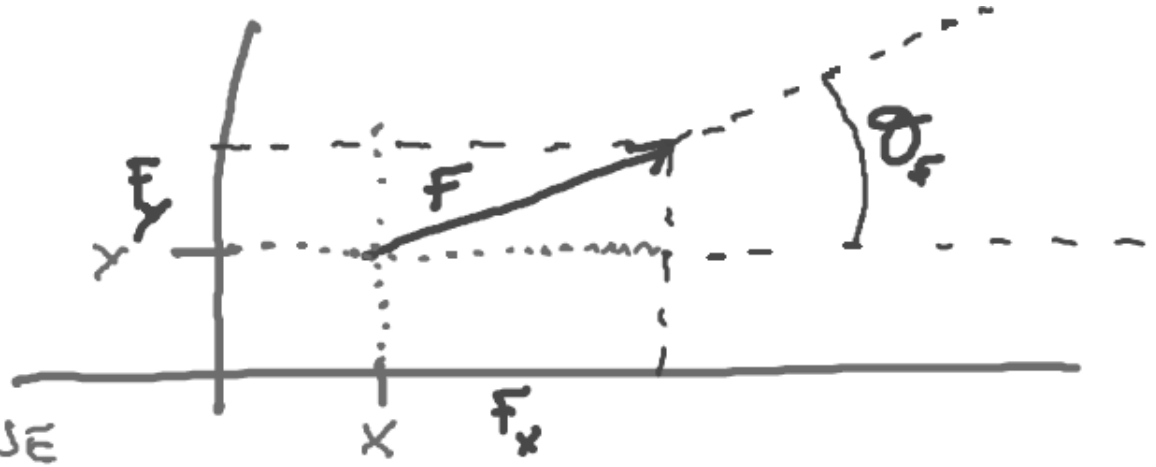
Struttura Dati Vettore Spostamento

(x, y) punto di applicazione

f modulo

ϑ_f direzione e verso

RAPPRESENTAZIONE

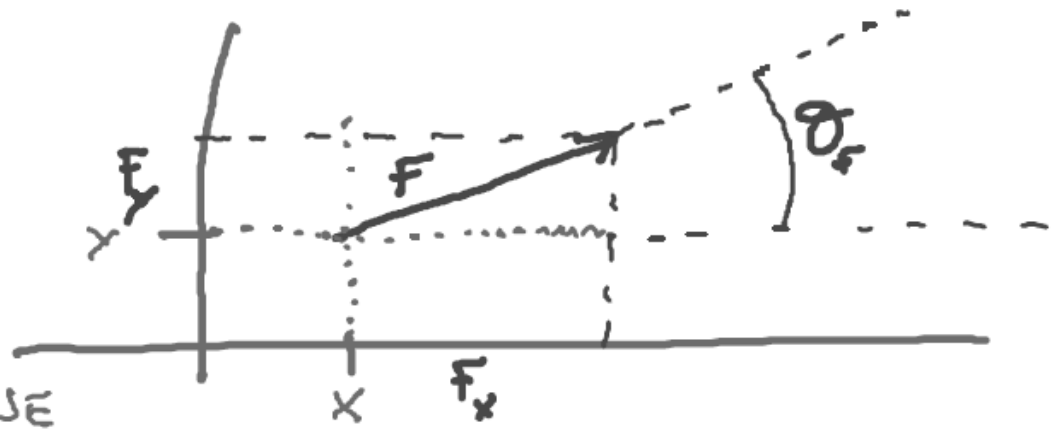


Struttura Dati Vettore SPostamento

(x, y) punto di applicazione

F modulo

ϑ_F direzione e verso



RAPPRESENTAZIONE

x	y	F	ϑ_F

variabile **vett**

double vett [4]

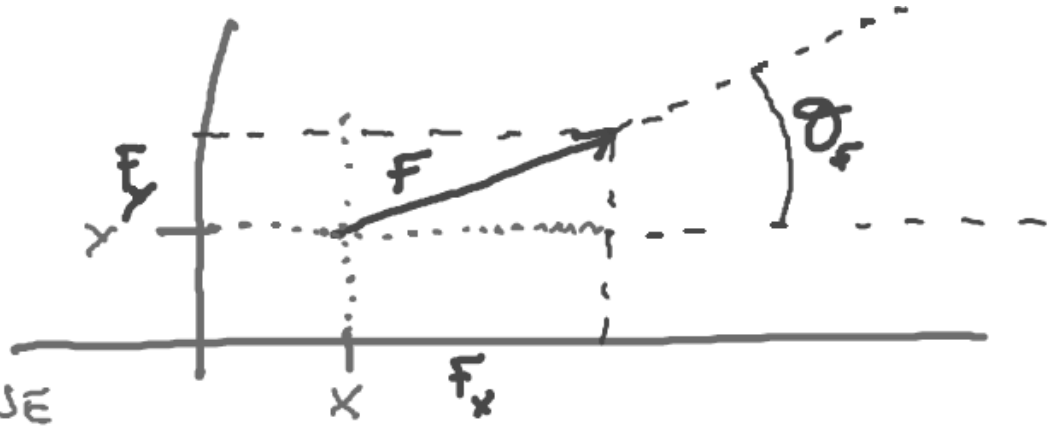
TIPO VettoreSpostamento ?

TIPI e COSTRUTTI di TIPO

(x, y) punto di applicazione

F modulo

ϑ_F direzione e verso



RAPPRESENTAZIONE

variabile **vett**
double vett [4]

$$F = \sqrt{F_x^2 + F_y^2}$$

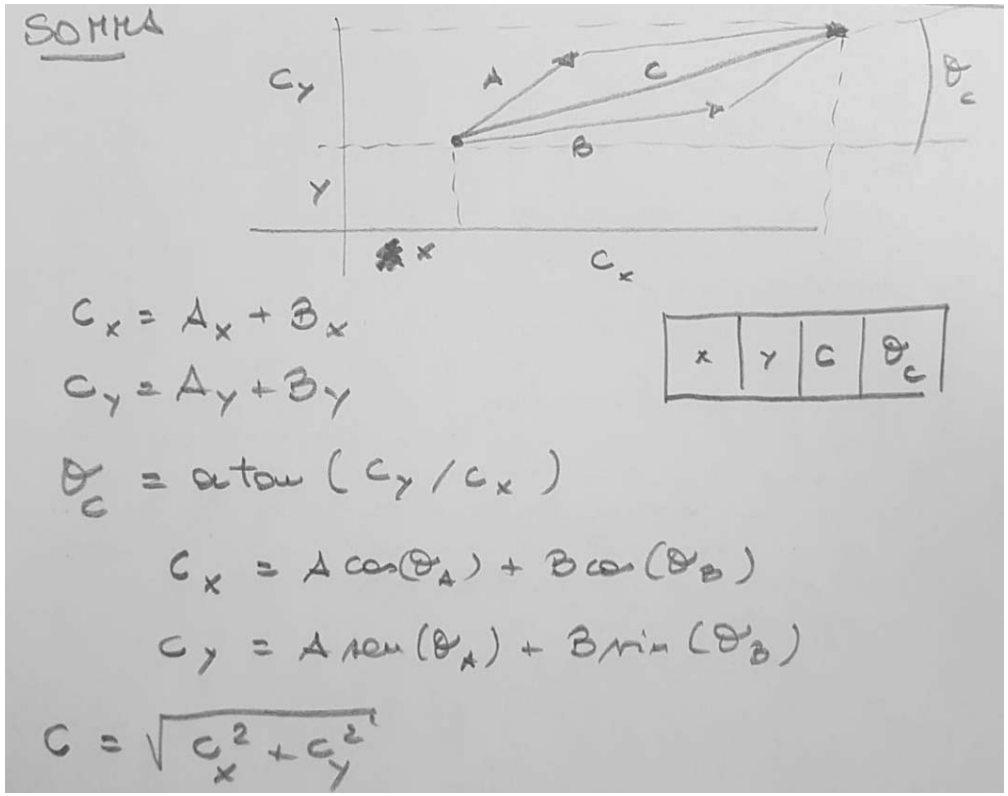
$$F_x = F \cdot \cos(\vartheta_F)$$

$$F_y = F \cdot \sin(\vartheta_F)$$

$$\tan(\vartheta_F) = \frac{F_y}{F_x}$$

$$\vartheta_F = \arctan(F_y / F_x)$$

somma di due vettori spostamento



☺

la funzione
RICEVE due vettori spostamento (v1 e v2) e un altro vettore spostamento da riempire (vSomma)

ed inserisce in vSomma i dati relativi ial vettore spostamento ottenuto come somma di v1 e v2

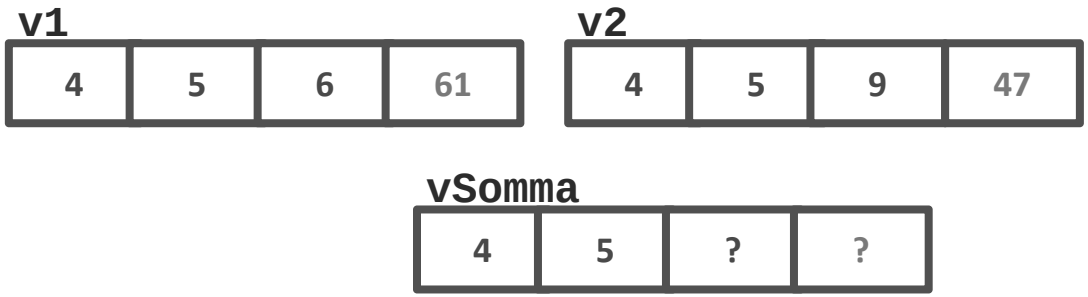
SCRIVERE i parametri formali

SCRIVERE il codice della funzione, in base alle formule in figura.

Solo dopo aver finito, guardare la slide successiva

somma tra vettori spostamento

```
void sommaVettSpost (          ☺          ) {
 double risX, risY;
```

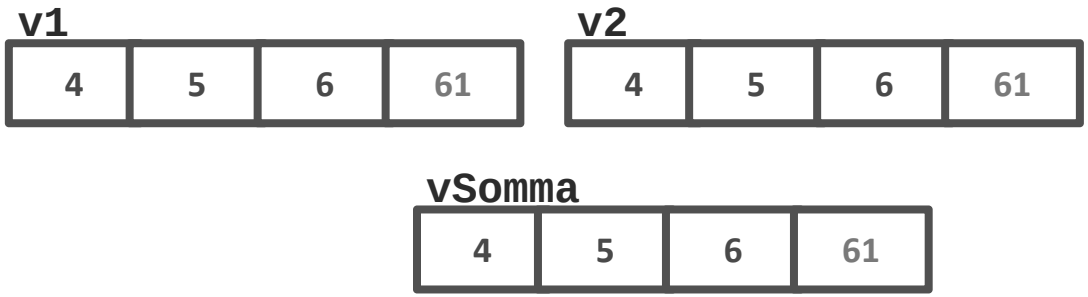


☺

```
return;
}
```

somma tra vettori spostamento

```
void sommaVettSpost (double v1[4], double v2[4],
                                   double vSomma[4]) {
 double risX, risY;
```




```
 vSomma[0] = v1[0];

 risX =   v1[2]*cos(v1[3]) + v2[2]*cos(v2[3]);

 vSomma[2] = sqrt(risX*risX +          ); /* il modulo */

 vSomma[3] =                              /* theta */
return;
}
```

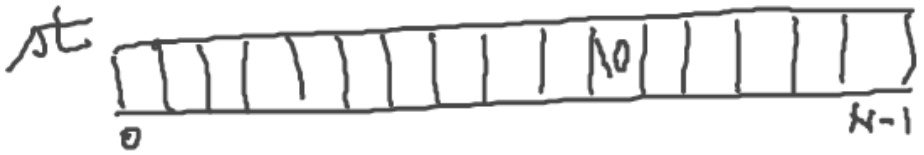
Stringhe

RAPPR ARRAY DI **CHAR** + CONVENZIONE

`char st[N]`

- sequenze di al più N-1 caratteri alfanumerici SIGNIFICATIVI
- NULL-TERMINATE

CARATTERE **NULL** ≡ '\0'



FONZ. funzioni nella string.h : I/O, copia, analisi...

TIPO STRINGA ?

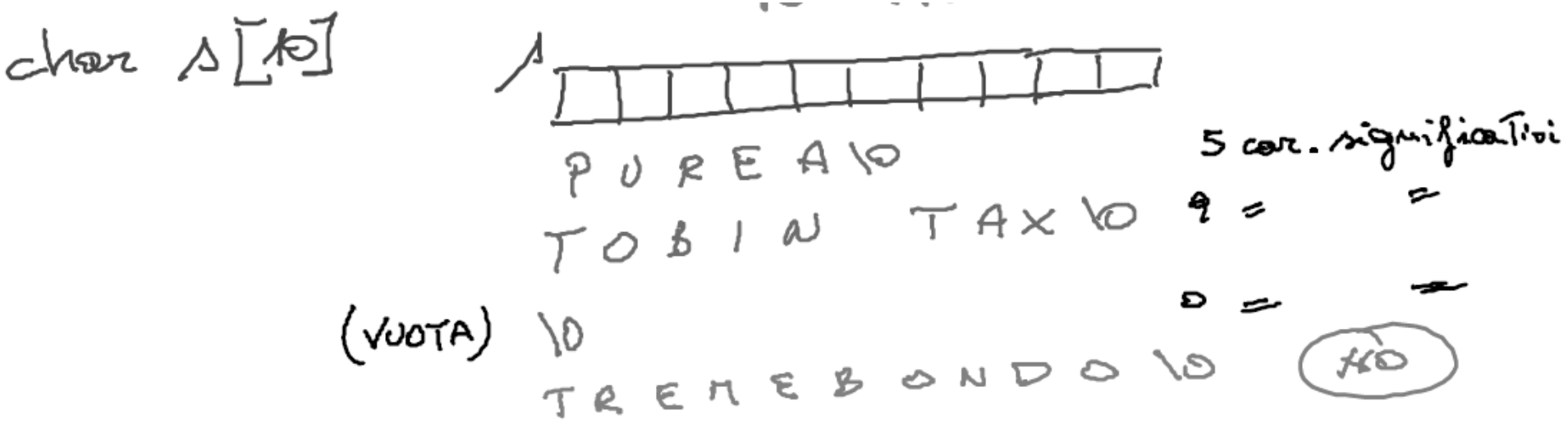
char *

char []

la libreria che contiene molte funzioni per gestire stringhe (definite come array di caratteri null terminati)
e` la **`string.h`**

Stringhe – cosa puo` essere rappresentato

N=10



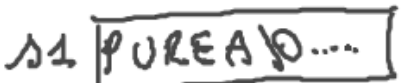
Alcune funzioni in string.h: strcpy()

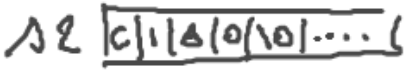
string copy

```
void strcpy (char *, char *)
             (1)     (2)
```

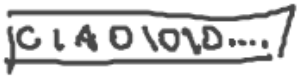
- copia i caratteri da (2) in (1) (significativi)
- aggiunge '\0' dopo l'ultimo

QUINDI s1 DEVE ESSERE UN ARRAY SUFFICIENTEMENTE LUNGO

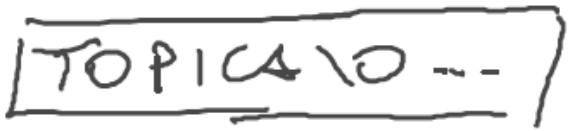




strcpy (s1, s2)

dopo s1 

strcpy (s1, "TOPICA")

dopo s1 

COSTANTE STRINGA

Alcune funzioni in string.h: strcmp()

string compare

```
int strcmp (char *, char *)
```

```
char p[...],
     s[...];
strcmp(p, s)
```

↳ rest. un intero

- < 0 se p precede s nell'ordinamento lessicografico
- $= 0$ se p è UGUALE a s
- > 0 se s precede p

Alcune funzioni in string.h: strlen()

string length

int strlen (char *)

strlen (s)

↔ # caratteri significativi di s

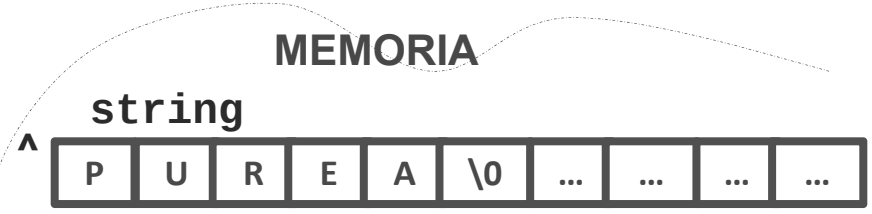
strlen("PUREA") ↔ 5

strlen("") ↔ ?

Simulazione di strlen() (come si fa a contare i caratteri in una stringa?)

chiamata `strLung (string);`

```
int strLung (char *);
int main {
  char string[31]; /* al + 30 signif.*/
  ...
  scanf("%s", string);
  printf("ho letto %s\n", string);
  printf("di lunghezza %d\n", strLung(string));
return 0;
}
```



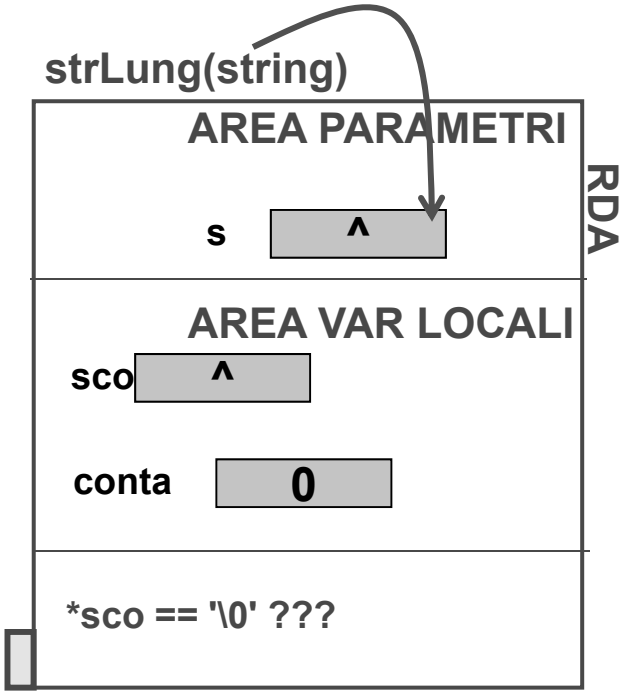
0) int conta, char * sco (char *s parametro formale)

1) INIT sco = ☺ conta = ☺

secondo passo … come sara` 2)?

2) mentre *sco != '\0'
   - conta ☺
   - ☺

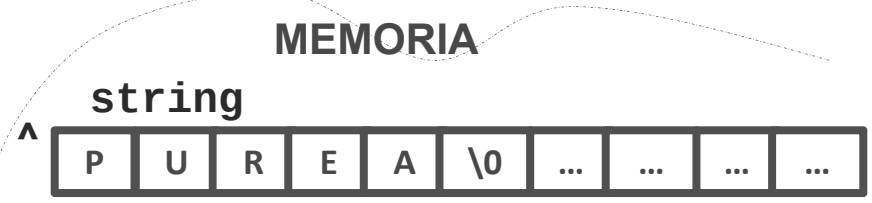
3) return conta



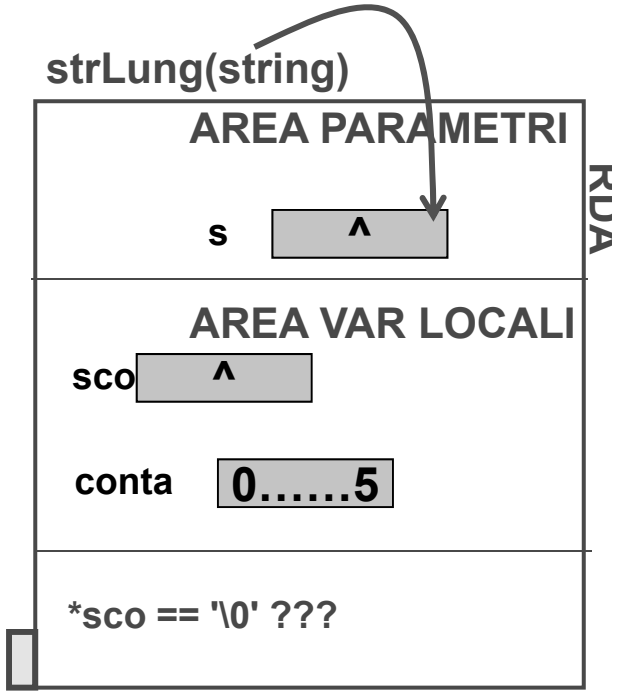
Simulazione di strlen() - Alg. per la funzione strLung()

chiamata `strLung (string);`

```
int strLung (char *);
int main {
  char string[31]; /* al + 30 signif.*/
  ...
  scanf("%s", string);
  printf("ho letto %s\n", string);
  printf("di lunghezza %d\n", strLung(string));
return 0;
}
```



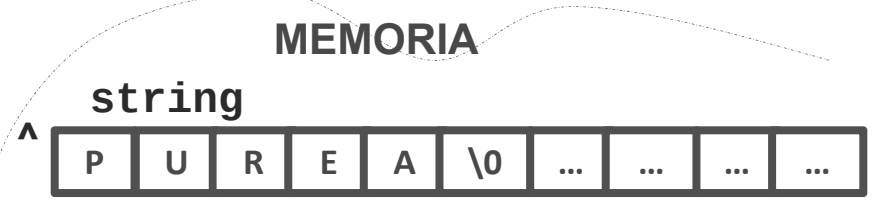

```
int strLung ( ☺ ) {

variabili locali

passo 1

passo 2




return conta;
}
```



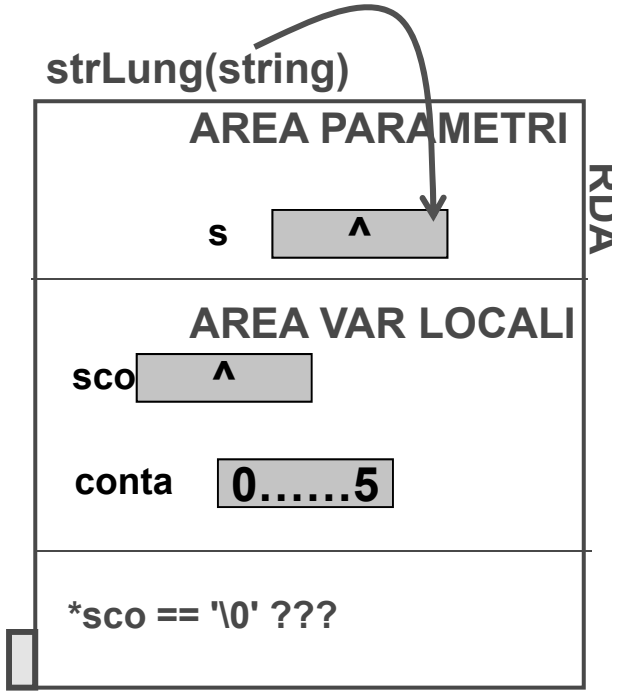
Simulazione di strlen() - Alg. per la funzione strLung()

chiamata `strLung (string);`

```
int strLung (char *);
int main {
  char string[31]; /* al + 30 signif.*/
  ...
  scanf("%s", string);
  printf("ho letto %s\n", string);
  printf("di lunghezza %d\n", strLung(string));
return 0;
}
```

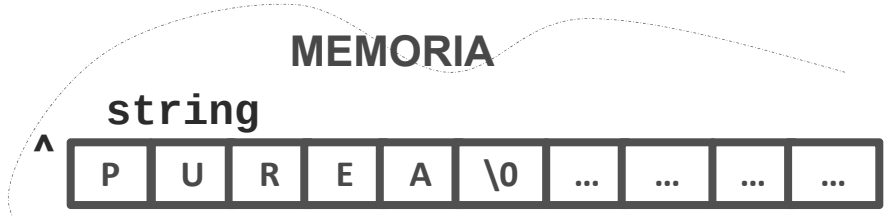



```
int strLung (char *s) {
 char *sco;    int conta;

 sco = s;   conta = 0;

 while (*sco!='\0') {
   conta++;
   sco++;
 }
return conta;
}
```



Variazioni … prima …

```
int strLung (char *s) {
 char *sco;    int conta;

 sco = s;     conta = 0;

 while (*sco!='\0')
   conta++;
   sco++;
 }
return conta;
}
```



(*sco != '\0') e (*sco)

sono espressioni equivalenti

	*sco != '\0'	*sco
*sco DIVERSO DA NULL	1	≠0
*sco UGUALE A NULL	0	0

VALORI DI VERITÀ

cambiamento

while (*sco) {

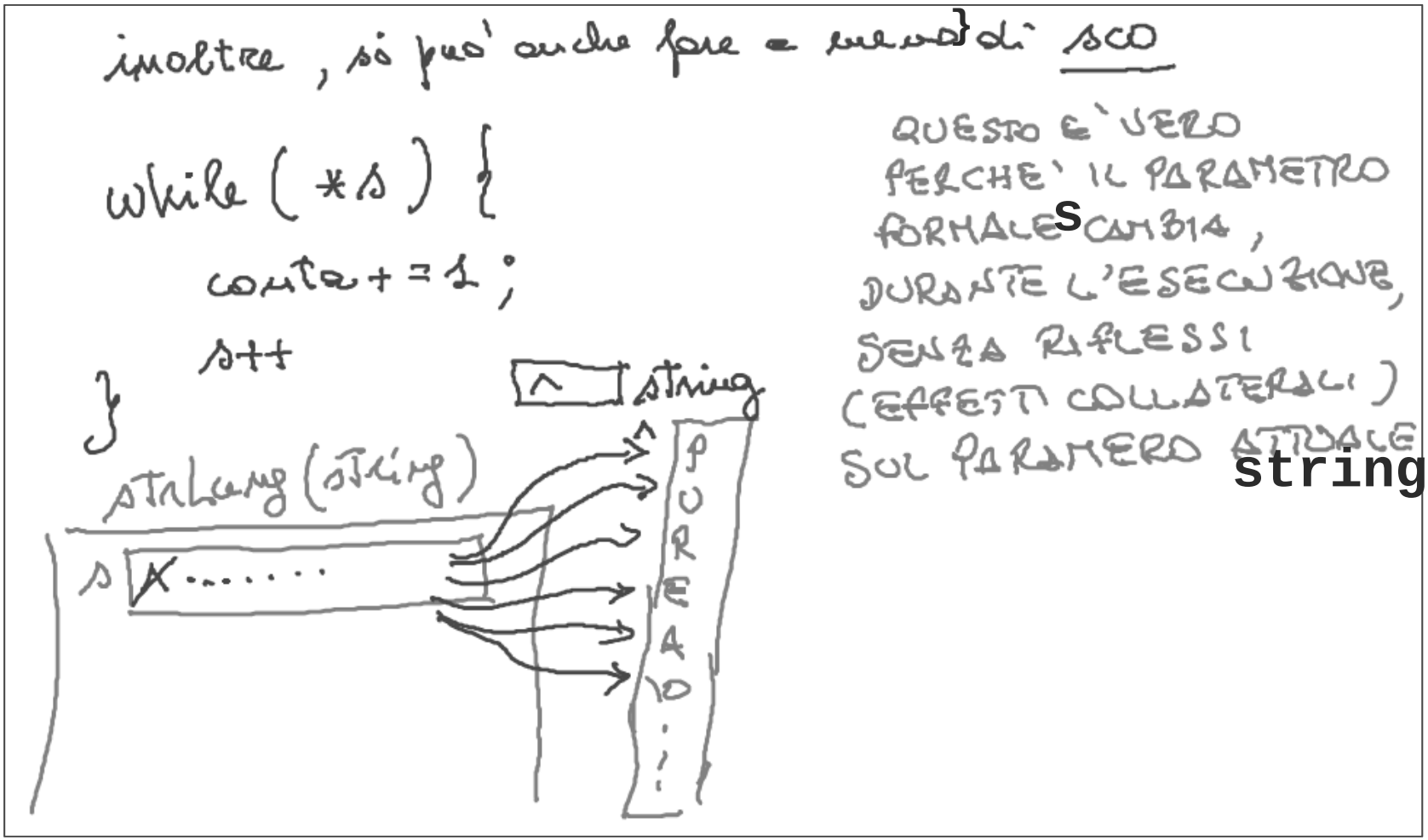
=

}

Variazioni … seconda …

```
int strLung (char *s) {
 char *sco;    int conta;

 sco = s;     conta = 0;
```

```
 while (*sco) {
   conta++;
   sco++;
  }
 return conta;
}
```



Variazioni … terza …

```
int strLung (char *s) {
  int conta;
 conta = 0;
 while (*s) {
   conta++;
   s++;
 }
return conta;}
```

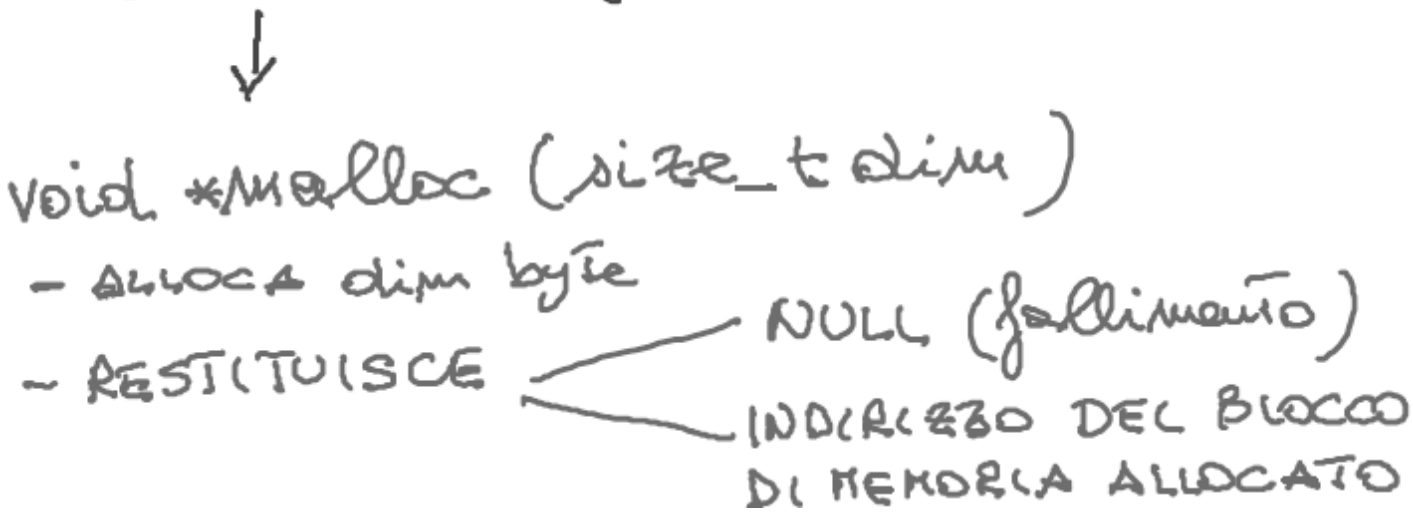
Allocazione Statica e dinamica…

ALLOCAZIONE STATICA

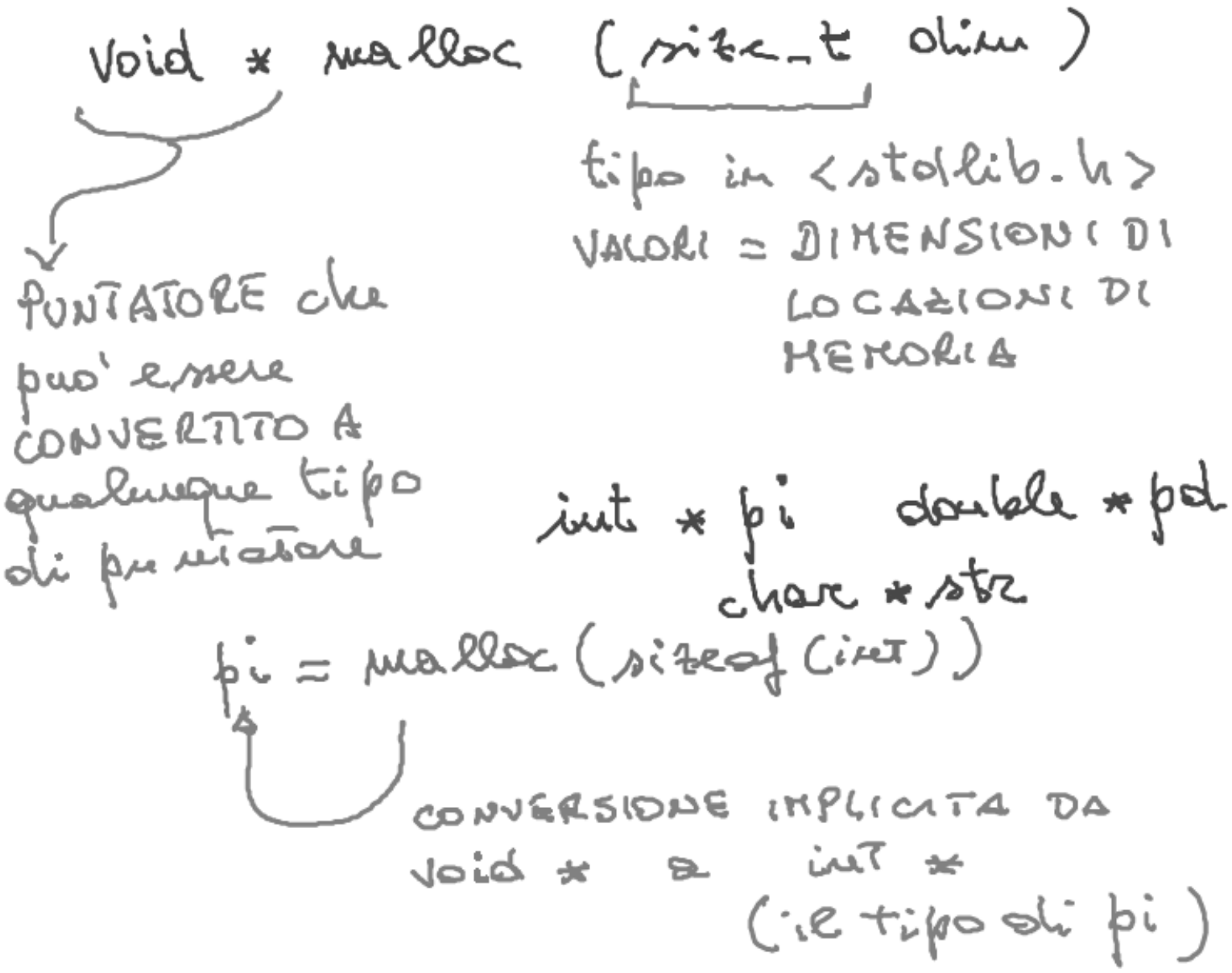
- decisa a compile-time
- in conseguenza di una DICHIARAZIONE

ALLOCAZIONE DINAMICA

- decisa a run-time
- in conseguenza di una ISTRUZIONE eseguita



malloc(): una funzione che restituisce un "puntatore generico



Esempi

```
int *pi
pi = malloc(sizeof(int));
```


```
double *pd = malloc(sizeof(double));
```

Esempi

```
char * str;

str = malloc(10 * sizeof(char));
```

blocco di 10 byte char

(array di char)

equivalente

```
str = malloc(10)
```

(sizeof(char) è 1)

Ecco a voi gli array dinamici

```
pi = malloc ( 5 * sizeof(int) );
```

blocco di 5 loc. interi

"array dinamico"

che succede se faccio

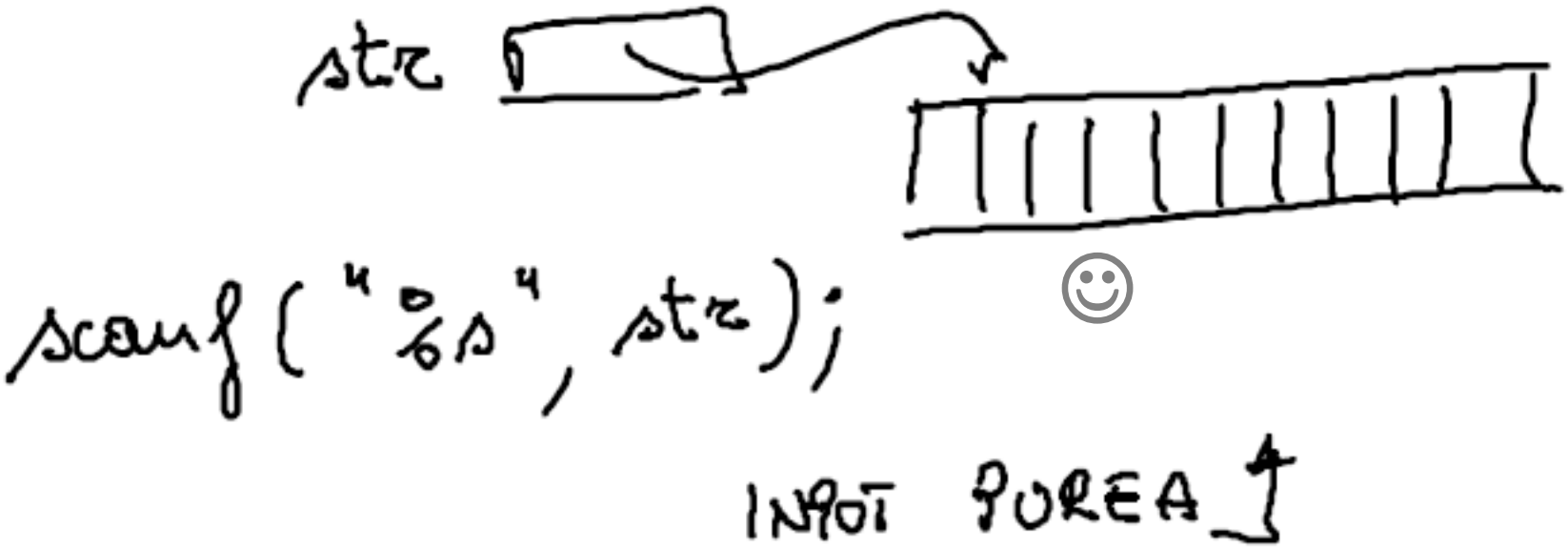
```
*pi = 6
*(pi+1) = 9
*(pi+3) = 18
```


```
for (i=0; i<5; i++)
    scanf("%d", pi+i)
```

INPUT 4, 20, 1, 47, 61


```
for (i=0; i<5; i++)
    scanf("%d", pi+1)
```

INPUT 4, 20, 1, 47, 61


str
scanf("%s", str);
INPUT PUREA

ma … l'allocazione era andata bene?

```
p = malloc(...)
/* CONTROLLO ALLOCAZIONE */
if (p == NULL)
    MESSAGGIO DI ERRORE
else  USO DEL BLOCCO
```

oppure, tradizionalmente...

```
if ( (p=malloc(...)) == NULL)
  MESSAGGIO ERRORE
else USO
```

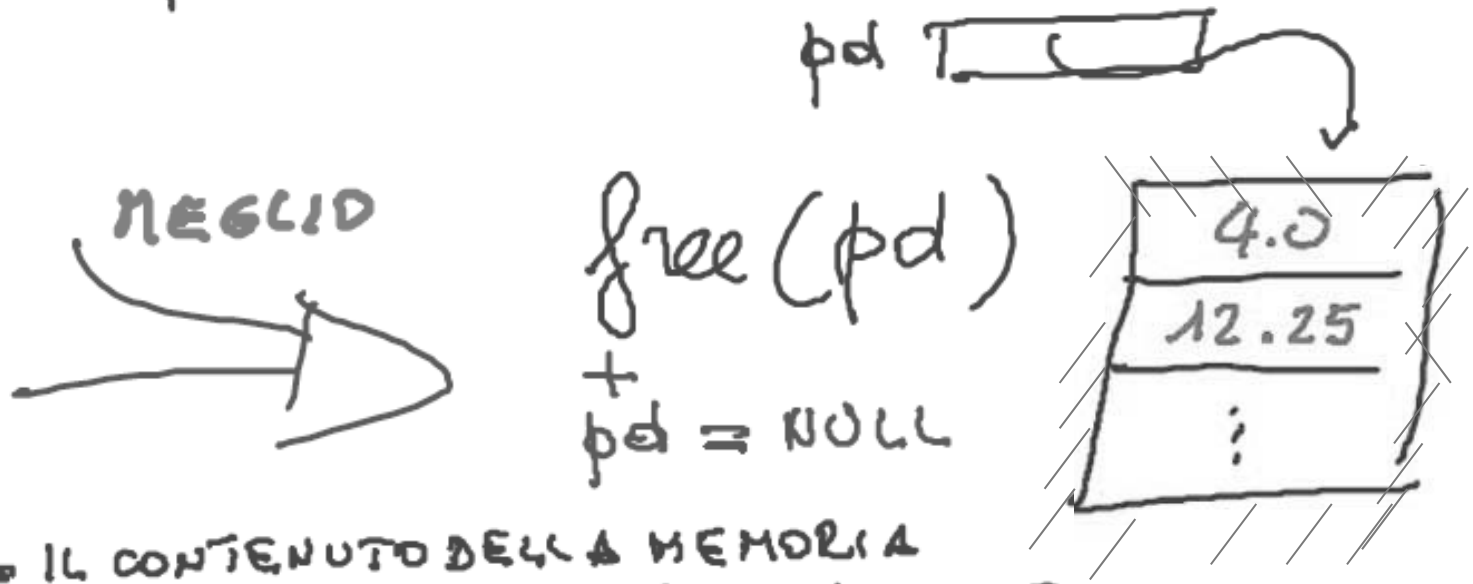
ok, ma se la memoria non serve piu`?

DEALLOCAZIONE --- NB

void free (void *punt)

LIBERA IL BLOCCO PUNTATO DA punt

- il puntatore non cambia contenuto …
- se punt == NULL non succede niente



- IL CONTENUTO DELLA MEMORIA DEALLOCATA NON VIENE CANCELLATO

Programma arrayDouble (EG5)

programma che memorizza in un array i double contenuti in un FILE TESTO. La dimensione dell'array deve essere "esatta" (cioe` uguale al numero di double da memorizzare)

1) lettura nome del file

2) apertura file

3) se file NON APERTO, messaggio e FINE
   altrimenti
   a) conta numero dati (lung) in file
   b) alloca blocco (array) di lung double
   c) ...

Programma arrayDouble (EG5) - schema

STRUTTURE DATI COINVOLTE

0)

```
char nomeFile[15]

FILE *fin

int lung=0
double dato
double * arrayDouble
```

per leggere e contare i dati nel file

1) lettura nome del file

2) apertura file

3) se file NON APERTO, messaggio e FINE
altrimenti
- **a) conta numero dati (lung) in file**
- **b)**
- **c) se NON ALLOCATO, messaggio, chiudi file e FINE**
  altrimenti
  - **file**
  - **lung**
- **d) close file**
- **e) stampalo**

4) FINE

Programma arrayDouble (EG5) – programma (1/2)

```
int main() {
      char nomefile[15];          FILE *fin;
      double *arrayDouble,        dato;
      int i,                      lung=0;


  scanf("%s", nomefile);
  if ((fin = fopen(nomefile, "r"))==NULL)


else {                    /* file aperto regolarmente */
        while (!feof(fin)) {



        }                 /*NB posizione parentesi*/
```

1) lettura nome del file

2) apertura file

3) se file NON APERTO, messaggio e FINE
   altrimenti

   a) conta numero dati (lung) in file

   b) alloca blocco (array) di lung double

   .....

Programma arrayDouble (EG5) – programma (2/2)

```
arrayDouble = malloc(lung*sizeof(double));
 if (arrayDouble==NULL)


 else {       rewind(fin);
        for(i=0; i<lung; i++)


        fclose(fin);   /* chiusura file */
  }
printf("il file e l'array contengono i seguenti %d elementi:\n", lung);
 if (arrayDouble)      /* se l'array non e` vuoto ... */



}      /* fine else “file aperto regolarmente” */
printf("FINE\n");
return 0;  }
```

b) alloca blocco (array) di lung double
c) se NON ALLOCATO, messaggio e FINE

altrimenti
- **rewind file**
- **lung letture dal file nel blocco**

d) close file

e) se il blocco era stato allocato stampalo

4) FINE