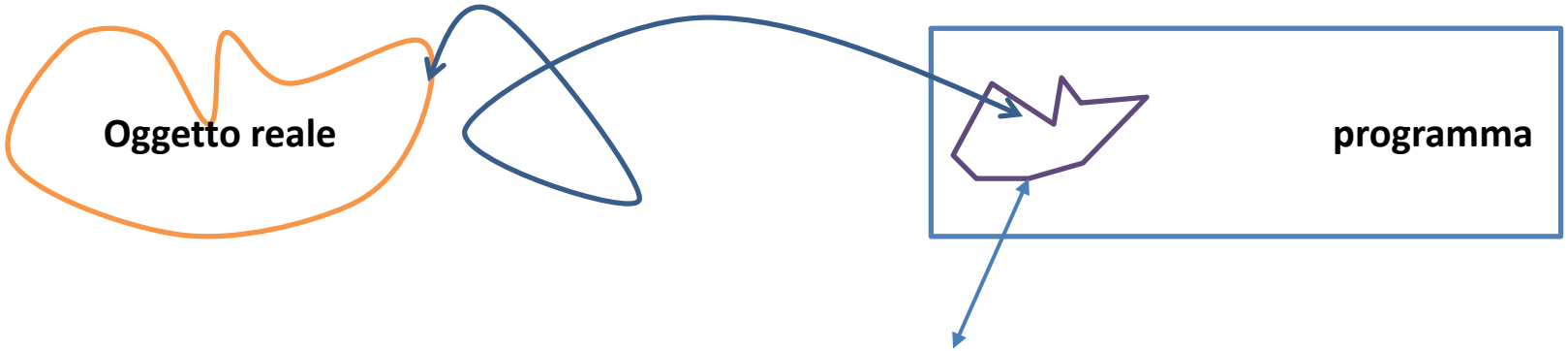
Tecniche della Programmazione, lez.15

- **Strutture dati meno elementary: vettore spostamento**
- **Un'altra struttura dati interessante: stringhe e relativa libreria**
- **Allocazione statica e dinamica; deallocazione**

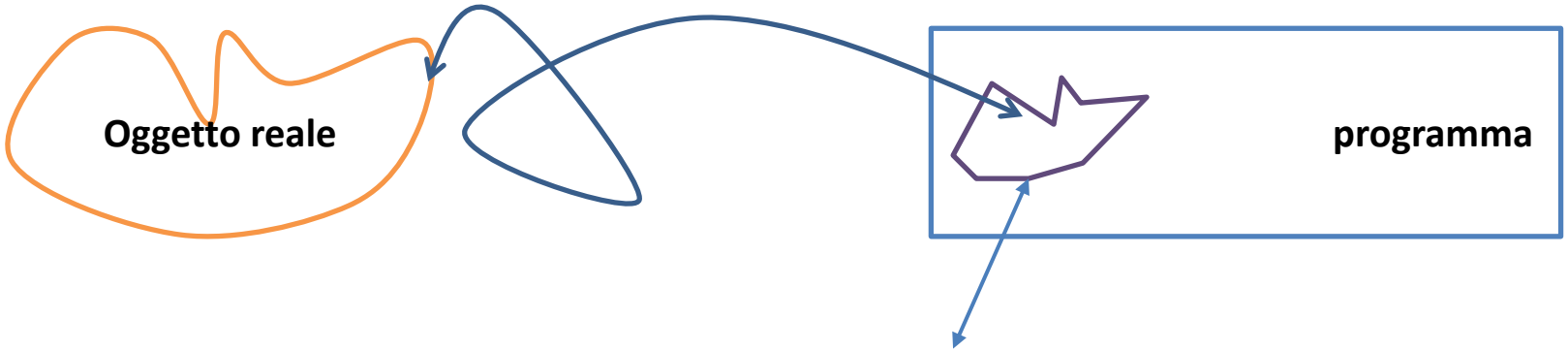
Risposta al problema di
- Rappresentare gli oggetti del problema (mondo reale) nel programma



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

Risposta al problema di

- Rappresentare gli oggetti del problema (mondo reale) nel programma, attraverso un'organizzazione propria dei dati
- Per permettere che la gestione dei dati simuli quella delle informazioni/oggetti reali



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

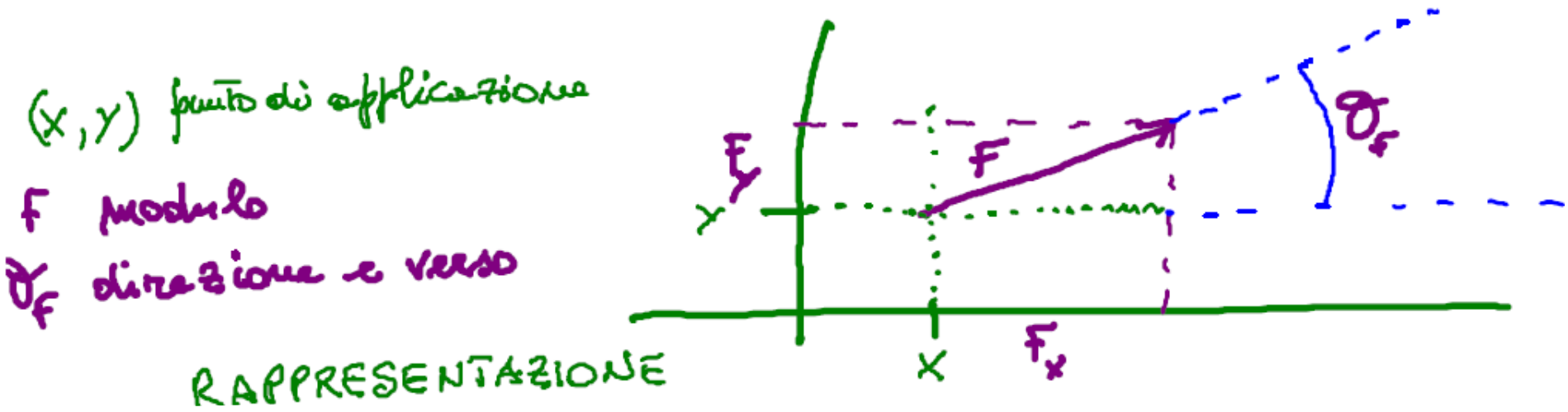
TIPO = { rappresentazione in memoria dei valori
funzionalità applicabili a quei valori

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



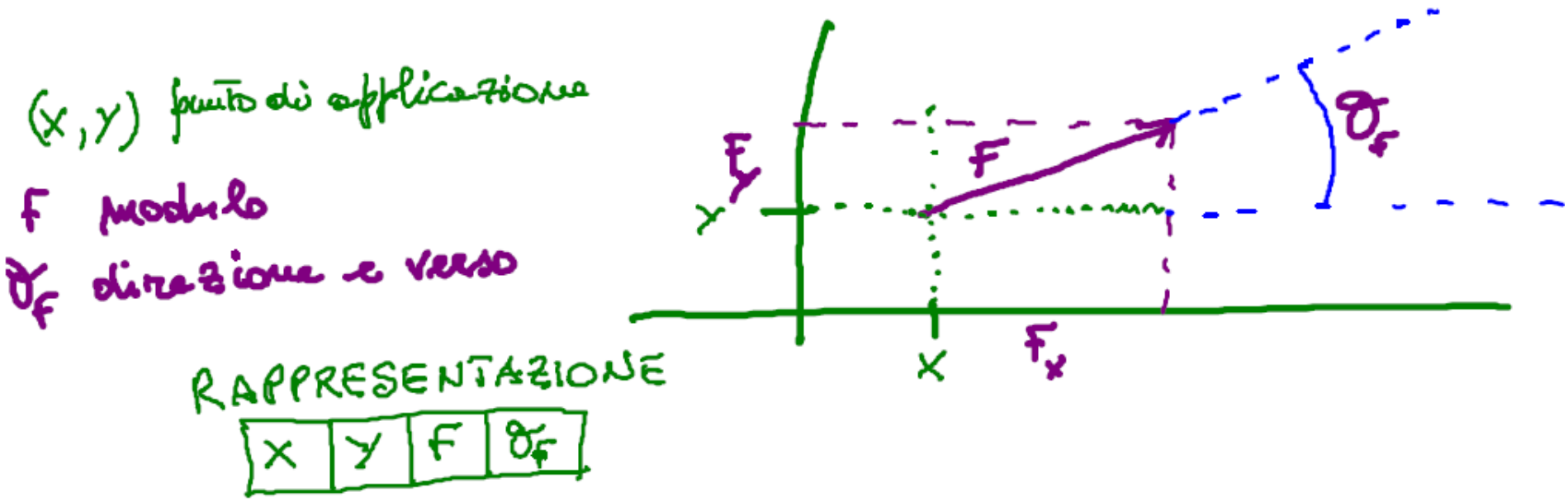
come rappresentare un vettore spostamento in un programma?

e` possibile fare in modo che un singolo vettore spostamento sia rappresentato nel programma come un singolo oggetto che contiene (memorizza) tutte le informazioni significative di un vettore spostamento

l'oggetto conterrebbe la "collezione di dati" che rappresenta le informazioni significative riguardo al vettore. Questa collezione potrebbe essere una collezione di variabili, o magari una sola variabile strutturata che contiene quelle variabili

cosi` , ad esempio, potremmo avere due vettori distinti, rappresentati come due diverse collezioni di variabiili, o due diverse variabili strutturate, e usarli, o passarli ad una funzione, per calcolare la loro somma.

Struttura Dati Vettore Spostamento



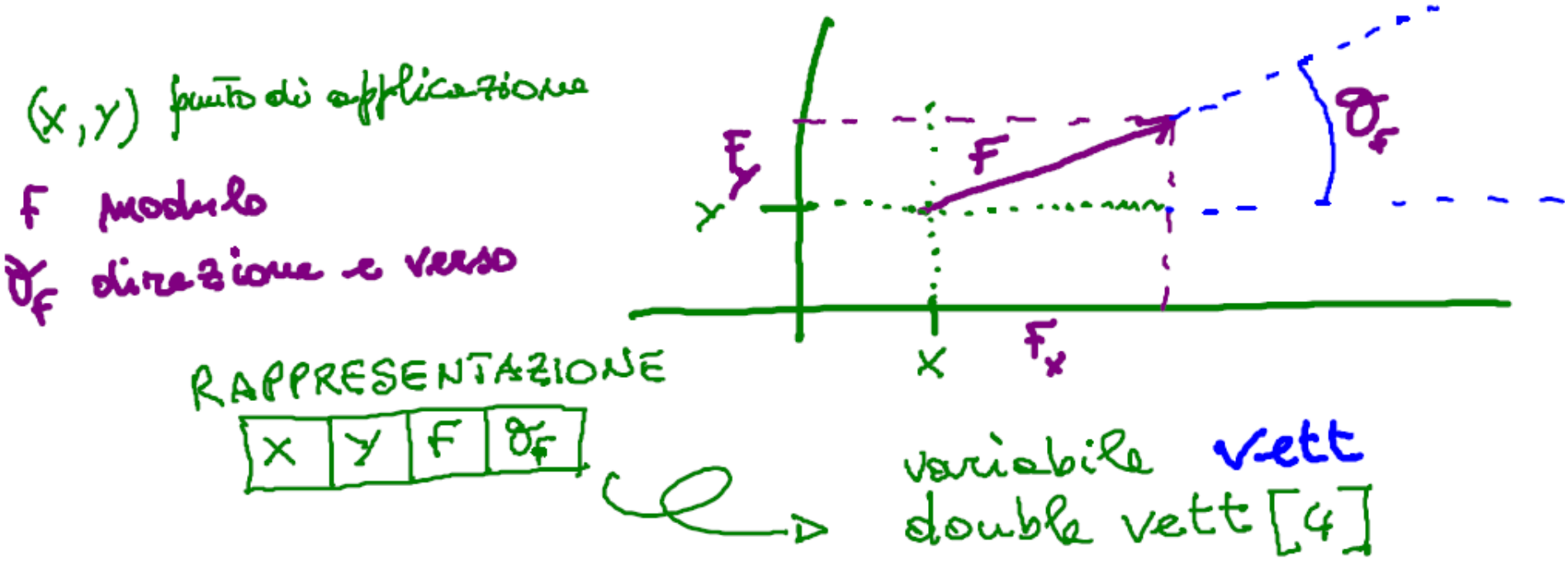
come rappresentare un vettore spostamento in un programma?

e` possibile fare in modo che un singolo vettore spostamento sia rappresentato nel programma come un singolo oggetto che contiene (memorizza) tutte le informazioni significative di un vettore spostamento

l'oggetto conterrebbe la "collezione di dati" che rappresenta le informazioni significative riguardo al vettore. Questa collezione potrebbe essere una collezione di variabili, o magari una sola variabile strutturata che contiene quelle variabili

cosi` , ad esempio, potremmo avere due vettori distinti, rappresentati come due diverse collezioni di variabiili, o due diverse variabili strutturate, e usarli, o passarli ad una funzione, per calcolare la loro somma.

Struttura Dati Vettore Spostamento



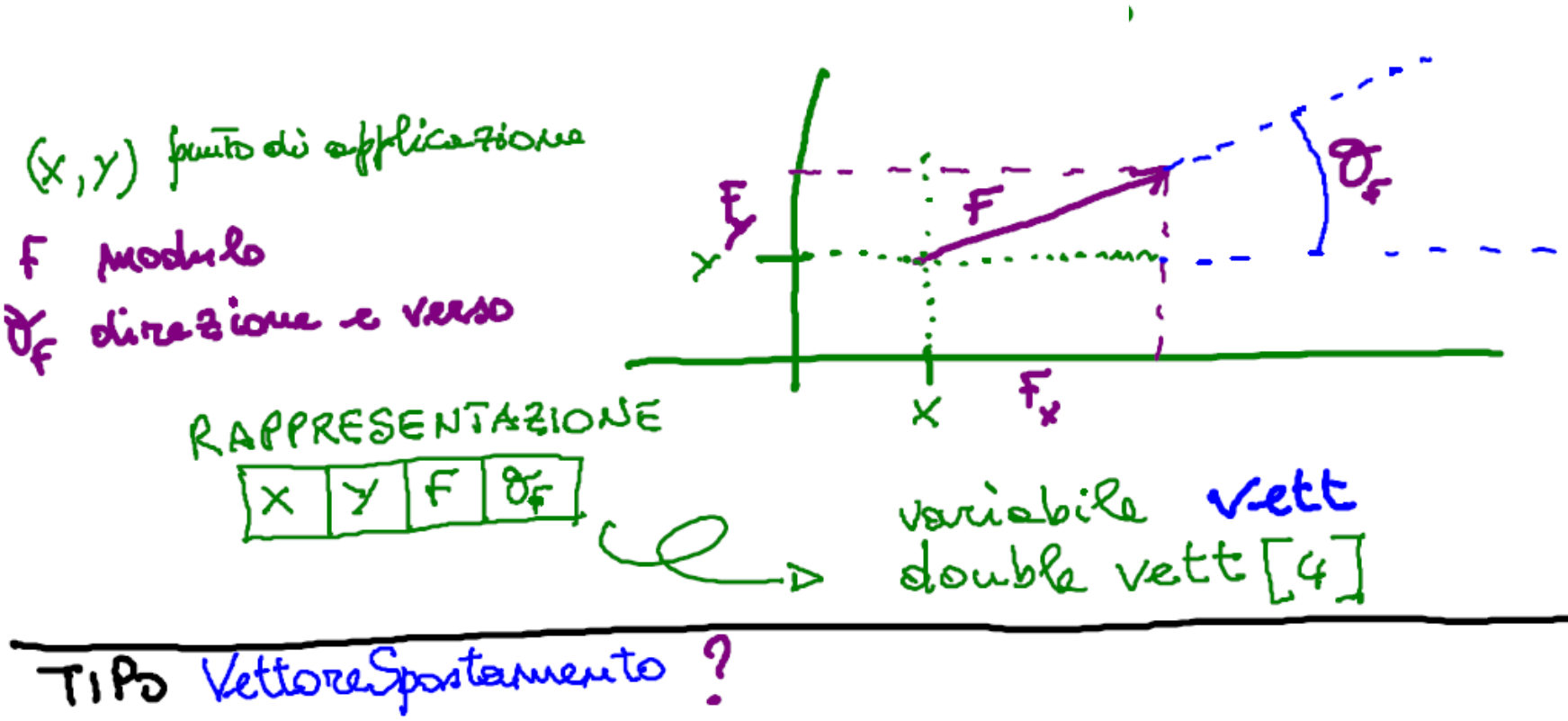
una possibile rappresentazione: array di double

la variabile vett rappresenta un vettore spostamento nel programma

ma se volessimo definire il TIPO VettoreSpostamento?

TIPO = struttura del singolo oggetto (valore) di quel tipo in memora,
e le funzioni che si possono usare su quel tipo di variabili …

Struttura Dati Vettore SPostamento



TIPO VettoreSpostamento ?

1) RAPPRESENTAZIONE: come e` fatto un valore di tipo VettoreSpostamento

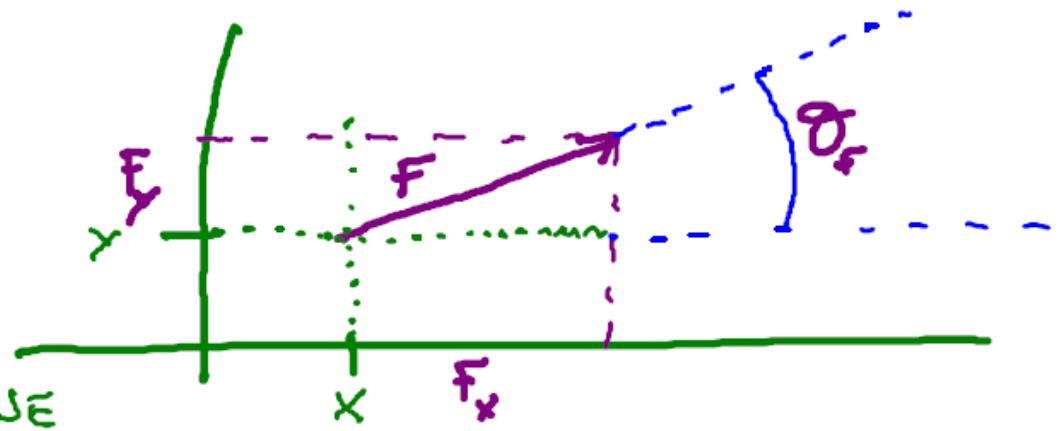
2) FUNZIONALITA`: somma, opposto, differenza, prodottoScalare, calcoloComponenti, calcoloTheta, calcoloModulo

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 ?

1) SD ✓✓

2) FUNZ: somma, opposto, diff, prodScalare, uguaglianza, calcoloComponenti, calcoloTheta, calcoloModulo

allora potremmo definire vet come una variabile di questo tipo

VettoreSpostamento vet;

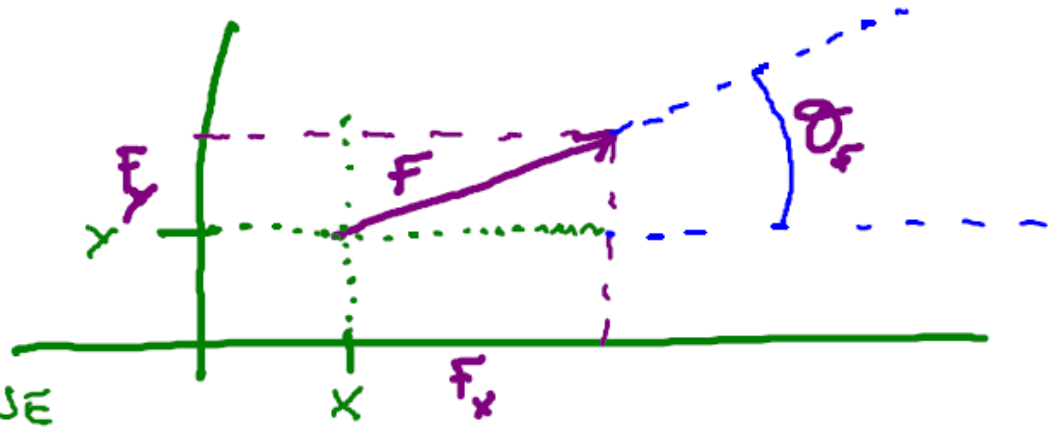
MA non lo facciamo adesso; non arriviamo adesso così lontano … ci accontentiamo di usare l'array vett per rappresentare un vettore spostamento, senza definire un TIPO

TIPI e COSTRUTTI di TIPO

(x, y) punto di applicazione

F modulo

ϑ_F direzione e verso



RAPPRESENTAZIONE

x	y	F	ϑ_F

variabile **vett**

double vett [4]

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

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

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

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

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

comunque, anche se non useremo il TIPO VettoreSpostamento, proviamo a definire qualche funzione che potrebbe far parte di una per definire una

libreria per il tipo VettoreSpostamento

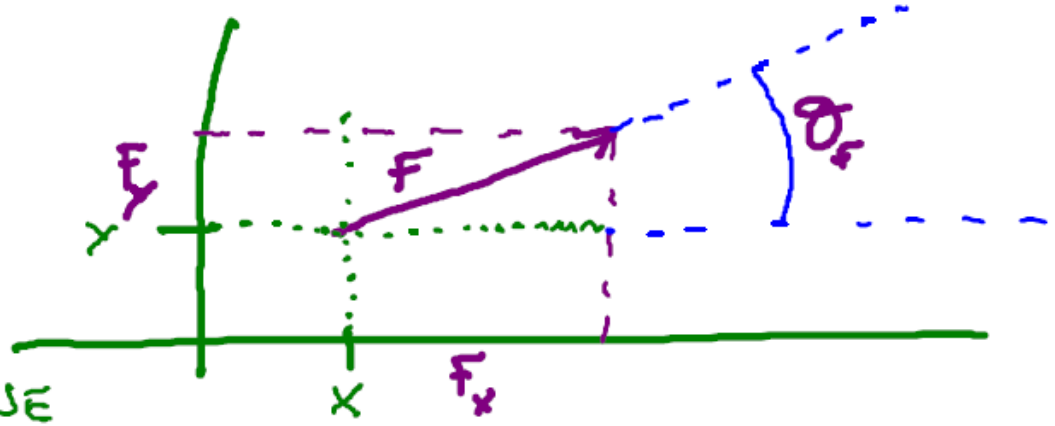
come rappresentiamo in memoria un vettore spostamento? Lo abbiamo detto: variabile array di quattro double.
Qualche funzione, su vettori spostamento cosi` rappresentati la possiamo definire ...

TIPI e COSTRUTTI di TIPO

(x, y) punto di applicazione

F modulo

ϑ_F direzione e verso



RAPPRESENTAZIONE

variabile **vett**

double vett [4]

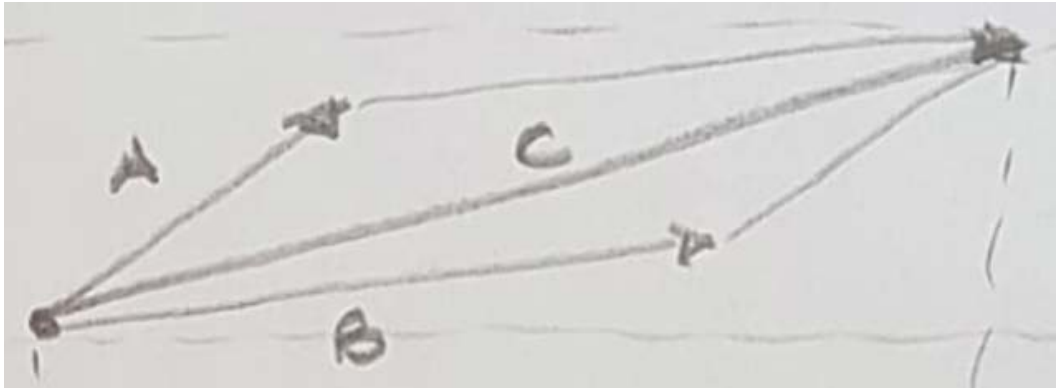
$$F = \sqrt{F_x^2 + F_y^2}$$

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

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

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

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



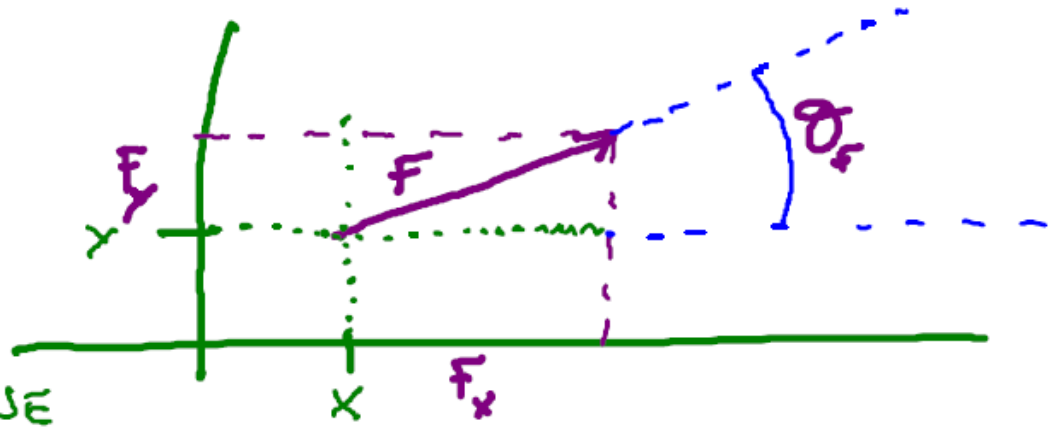
che variabili usiamo per questi tre vettoriSpostamento?
vettA, vettB, vettC … di che tipo?

TIPI e COSTRUTTI di TIPO

(x, y) punto di applicazione

F modulo

ϑ_F direzione e verso



RAPPRESENTAZIONE

x	y	F	ϑ_F

variabile **vett**

double vett[4]

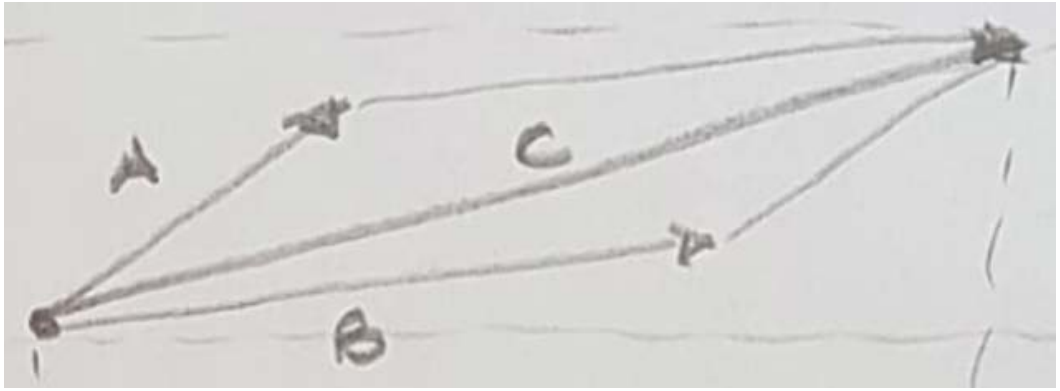
$$F = \sqrt{F_x^2 + F_y^2}$$

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

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

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

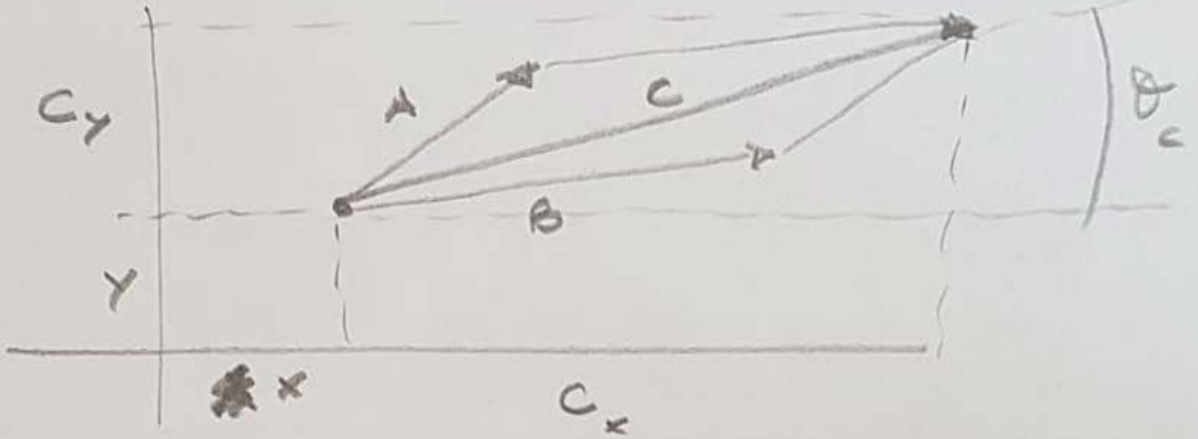
$$\vartheta_F = \arctan(F_y / F_x)$$




```
double vettA[4], vettB[4], vettC[4]
```

somma di due vettori spostamento

SOMMA



$$c_x = A_x + B_x$$

$$c_y = A_y + B_y$$

x	y	c	ϑ_c

$$\vartheta_c = \text{atan}\,(c_y / c_x)$$

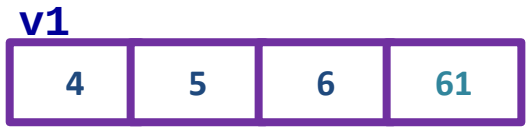
$$c_x = A\cos(\vartheta_A) + B\cos(\vartheta_B)$$

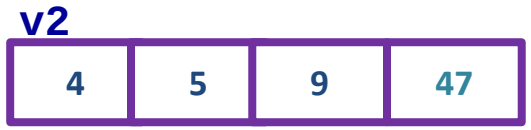
$$c_y = A\sin(\vartheta_A) + B\sin(\vartheta_B)$$

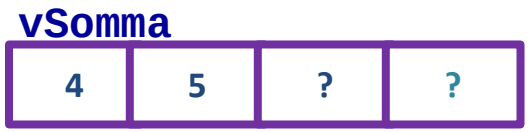
$$c = \sqrt{c_x^2 + c_y^2}$$

somma tra vettori spostamento

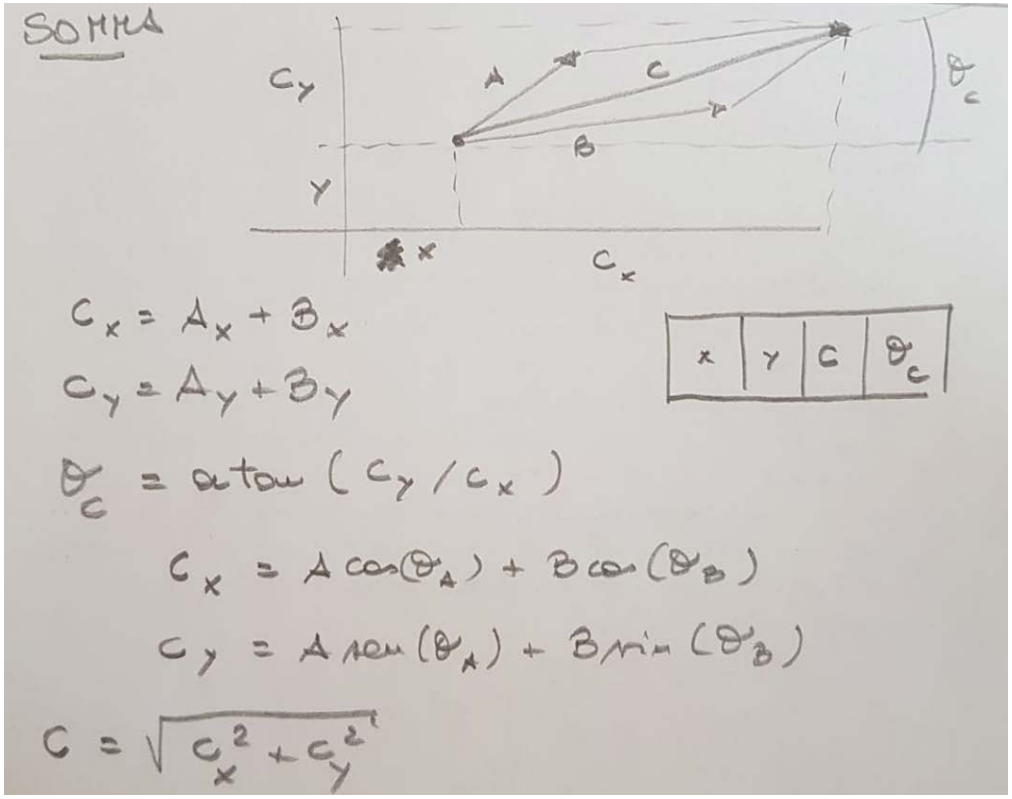
```
void sommaVettSpost (          ☺          ) {
 double risX, risY;
```







☺




```
return;
}
```

☺

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

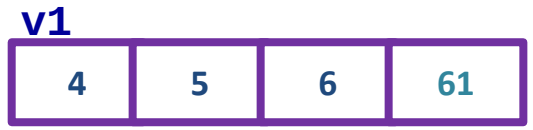
SCRIVERE i parametri formali

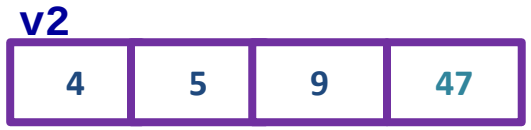
SCRIVERE il codice della funzione, in base alle formule in figura.

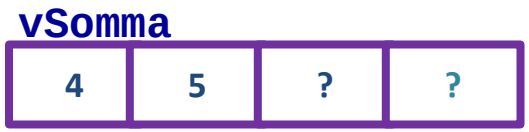
Solo dopo aver finito, guardare la slide successiva

somma tra vettori spostamento

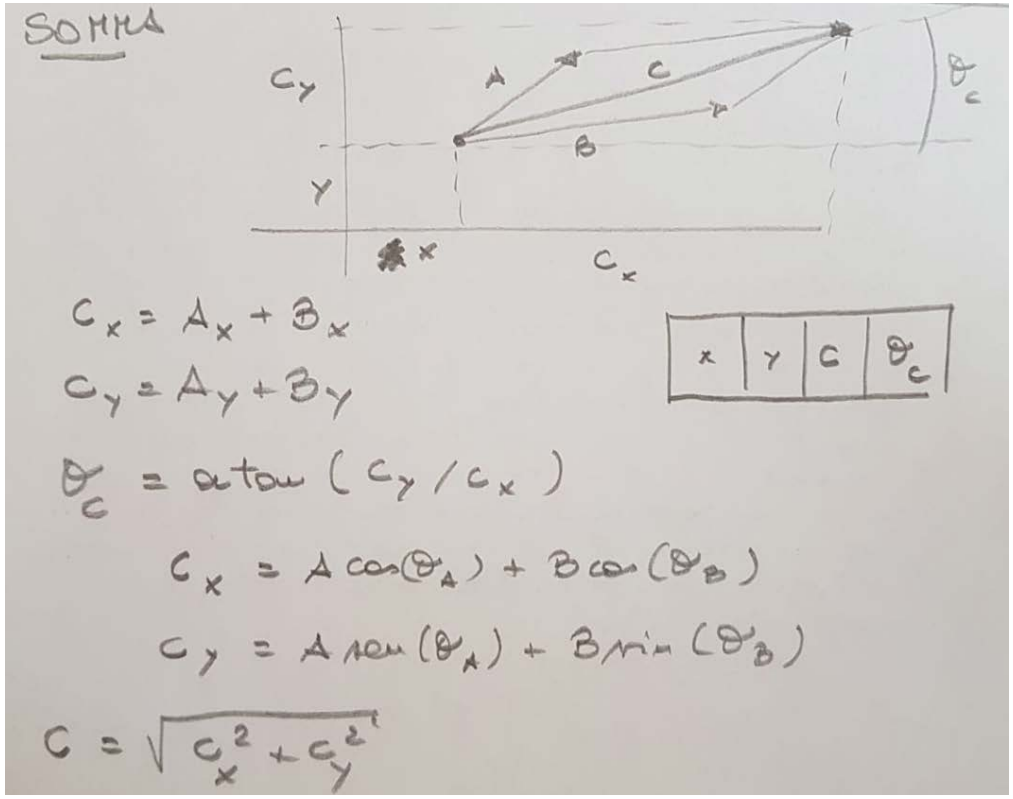
```
void sommaVettSpost (double v1[4], double v2[4],
                                   double vSomma[4]) {
 double risX, risY;
```







☺



☺

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

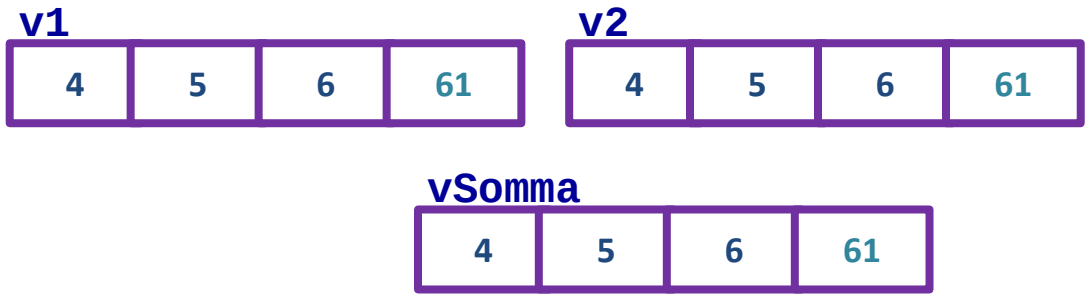
se non gia` fatto
SCRIVERE il codice della funzione, in base alle formule in figura.

Solo dopo aver finito, guardare la slide successiva

```
return;
}
```

somma tra vettori spostamento

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

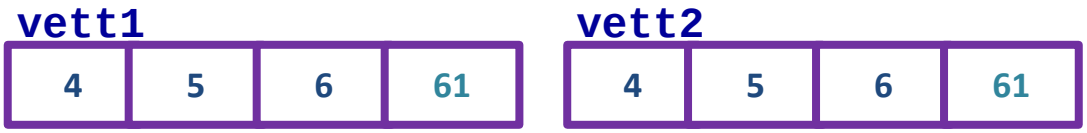



```
 vSomma[0] = v1[0];
 vSomma[1] = v1[1];   /* il punto di applicazione e` uguale per tutti */

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

 risY =   v1[2]*sin(v1[3]) + v2[2]*sin(v2[3]);

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

 vSomma[3] = atan (risY/risx);              /* theta */
return;
}
```

NB

Se avessimo un tipo . . . . . .TipoVettoreSpostamento
piu` o meno equivalente a . . . . . .double [4]
potremmo definire variabili come

```
TipoVettoreSpostamento vett1, vett2
```



per ora non possiamo …

Per definire tipi si definiscono
- SD (strutture dati per rappresentare i valori)
- FUNZ (funzionalita` per usare i valori di quel tipo)

tipicamente in LIBRERIE,

esempio (altro esempio): STRINGHE di caratteri

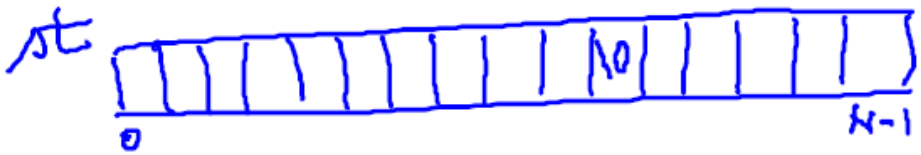
Stringhe

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

char s[10]

PUREA\0

5 car. significativi

TOBIN TAX\0

(VUOTA) \0

TREMEBONDO\0

Alcune funzioni in string.h: strcpy()

string copy

void strcpy (char * ① , char * ②)

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

QUINDI s1 DEVE ESSERE UN ARRAY SUFFICIENTEMENTE LUNGO

s1 | P | U | R | E | A | \0 | .... |

s2 | C | I | A | O | \0 | .... |

strcpy (s1, s2) dopo s1 | C | I | A | O | \0 | .... |

strcpy (s1, "TOPICA")

dopo s1 | T | O | P | I | C | A | \0 | -- |

↑ 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

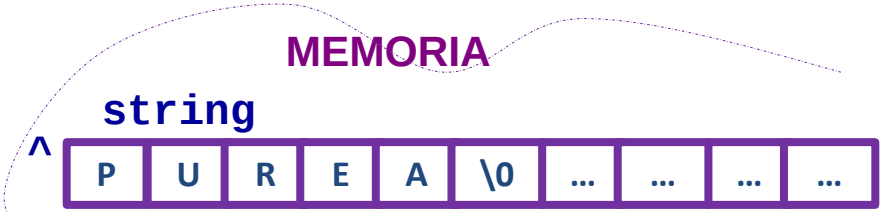
strlen("PUREA") ↪ 5

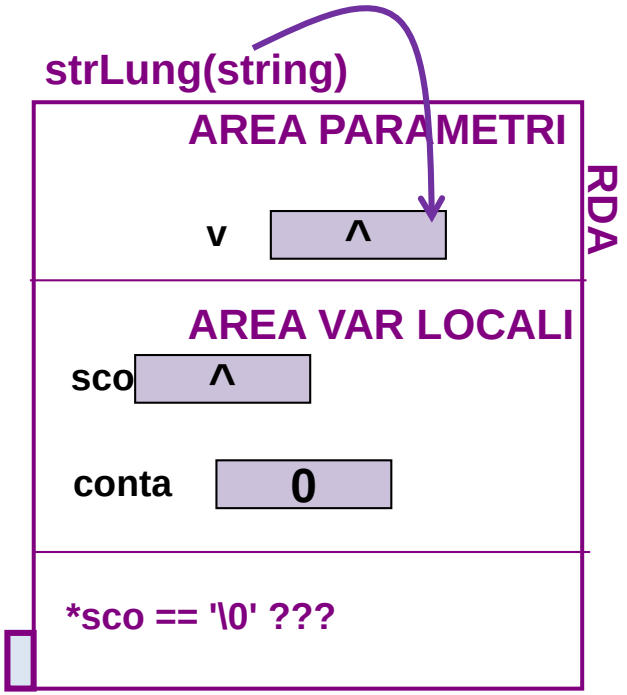
strlen(s) ↪ ?

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;
}
```





Simulazione di strlen()

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

chiamata `strLung (string);`

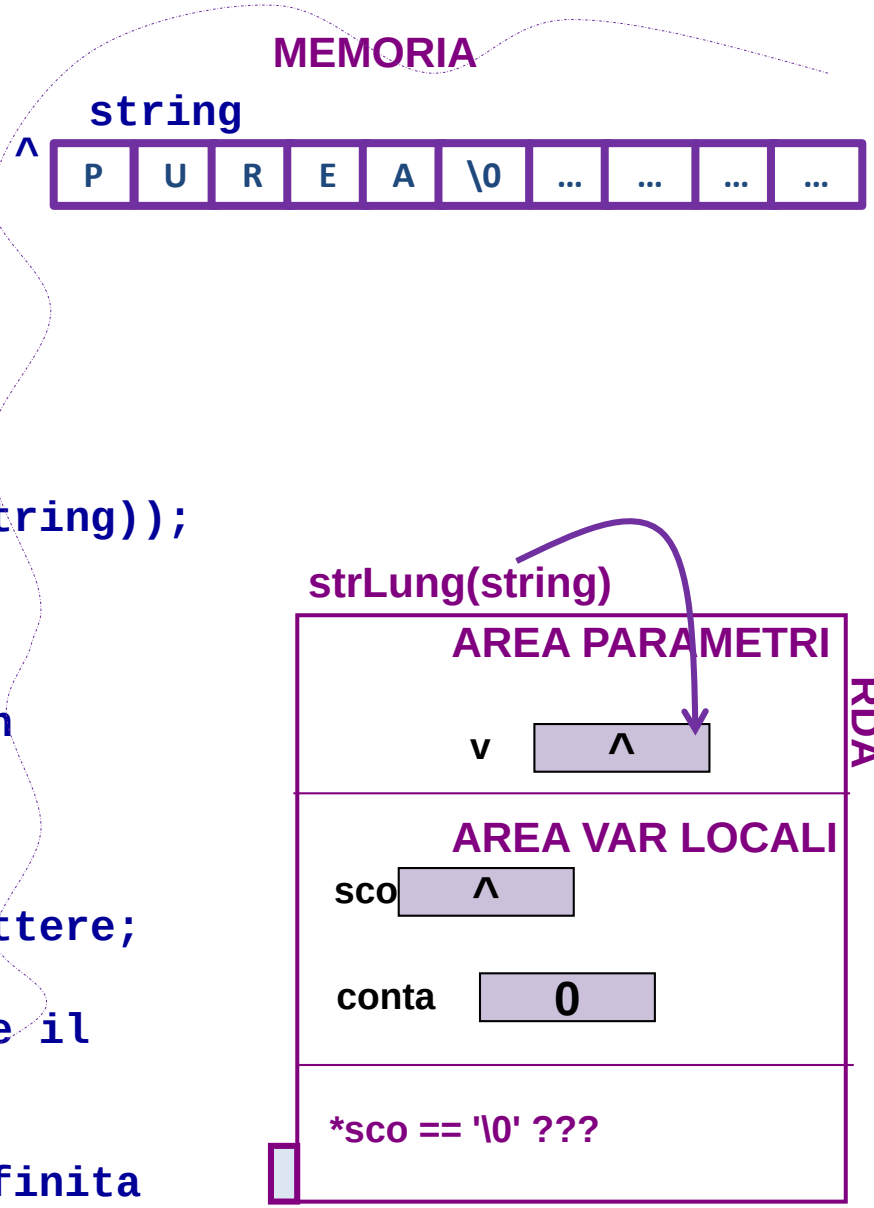
```
int strLung (char *);

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;
}
```

strLung usera` un contatore (conta) e un puntatore di scorrimento (sco);

durante la scansione della stringa, un carattere per volta, sco punta sul carattere;

il contatore aumenta di 1 ogni volta che il carattere e` significativo.

Quando sco punta sul carattere '\0' e` finita ...



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

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

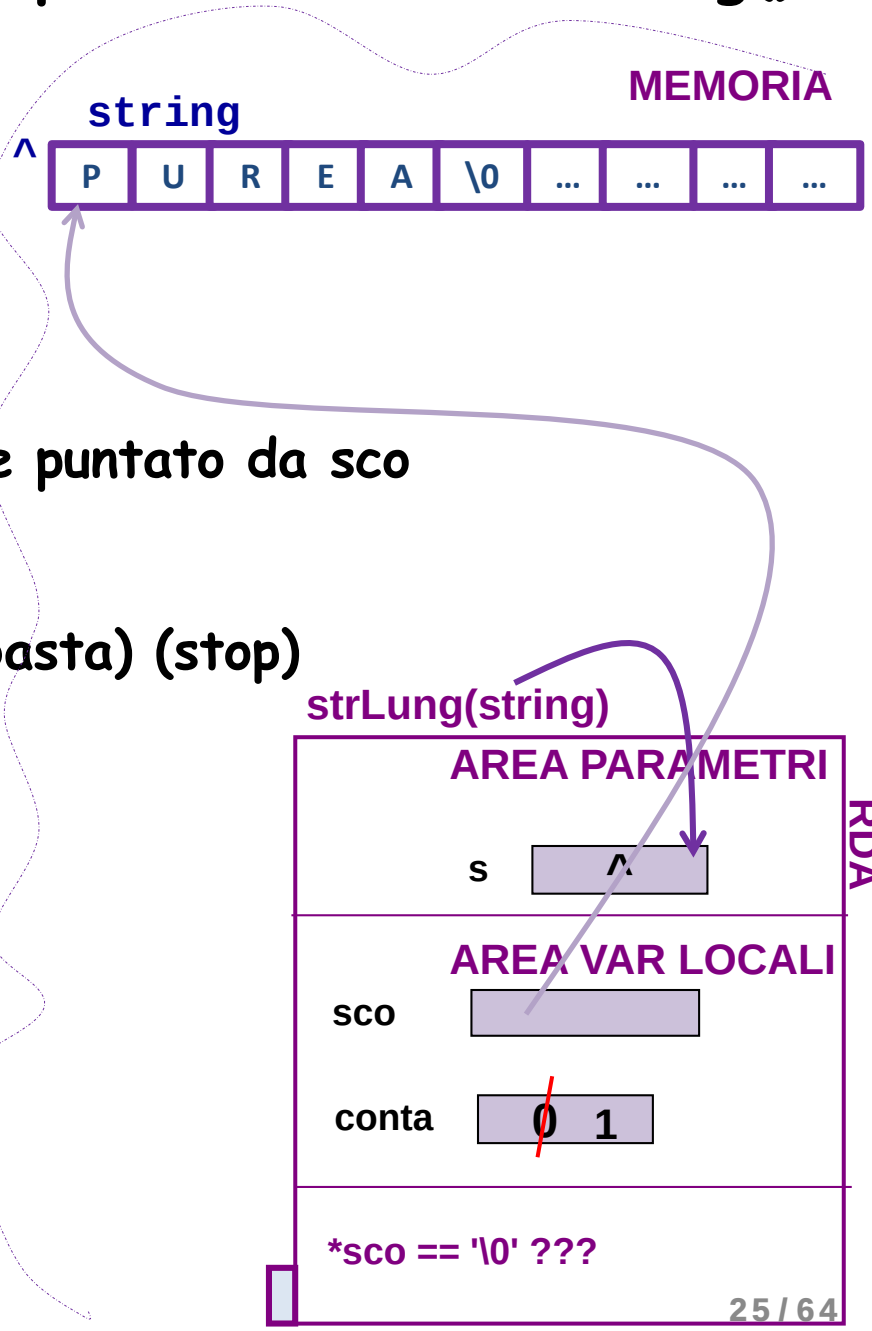
1) INIT sco = ☺ conta = ☺

secondo passo … come sara` 2)?
se *sco diverso da '\0' il carattere puntato da sco
e` ☺ quindi … conta ☺

else finita (stringa tutta scandita) (basta) (stop)

2) mentre *sco != '\0'
- conta ☺
- ☺

3) return conta



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

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

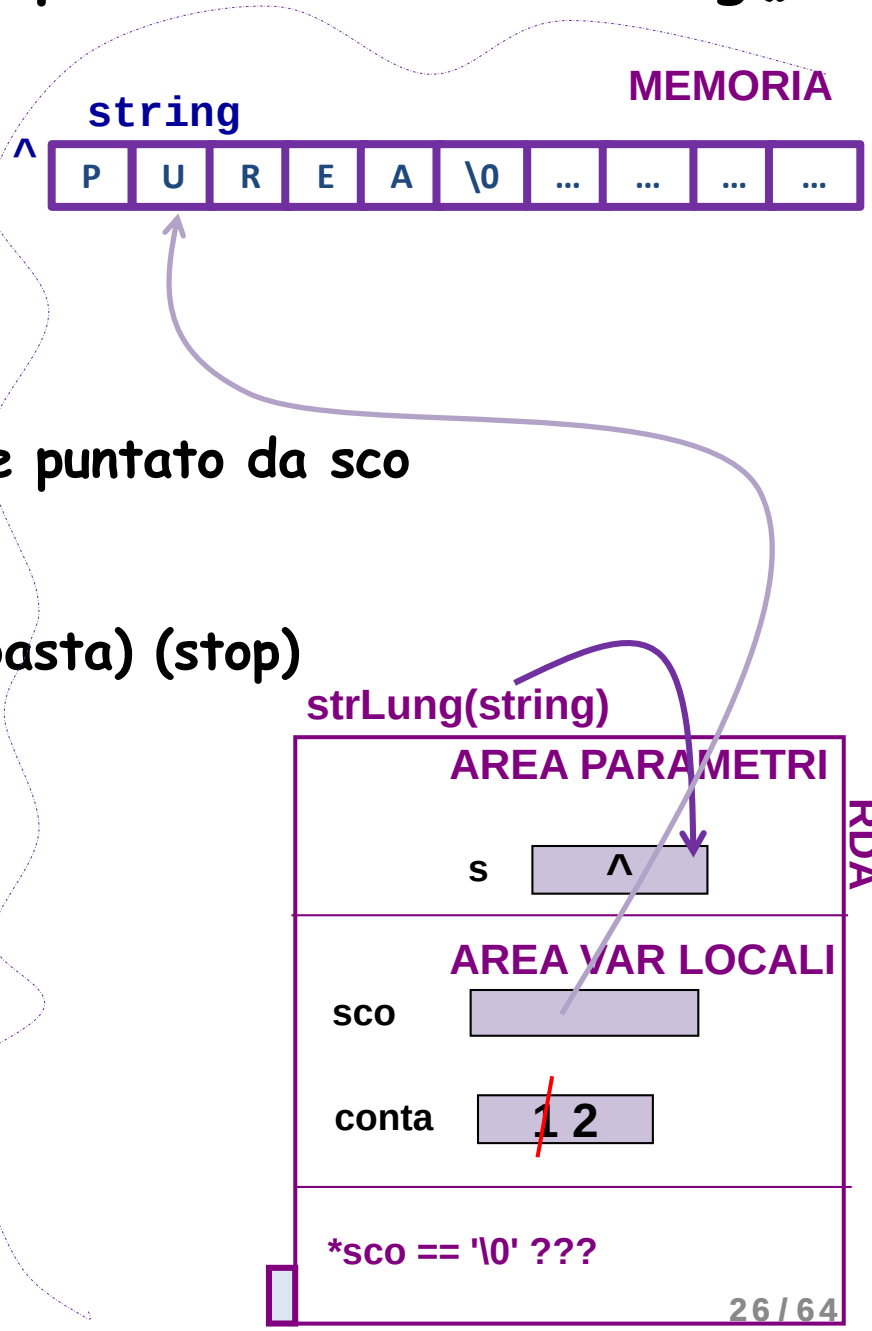
1) INIT sco = ☺ conta = ☺

secondo passo
se *sco diverso da '\0' il carattere puntato da sco
e` ☺ quindi … conta ☺

else finita (stringa tutta scandita) (basta) (stop)

2) mentre *sco != '\0'
- conta ☺
- ☺

3) return conta



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

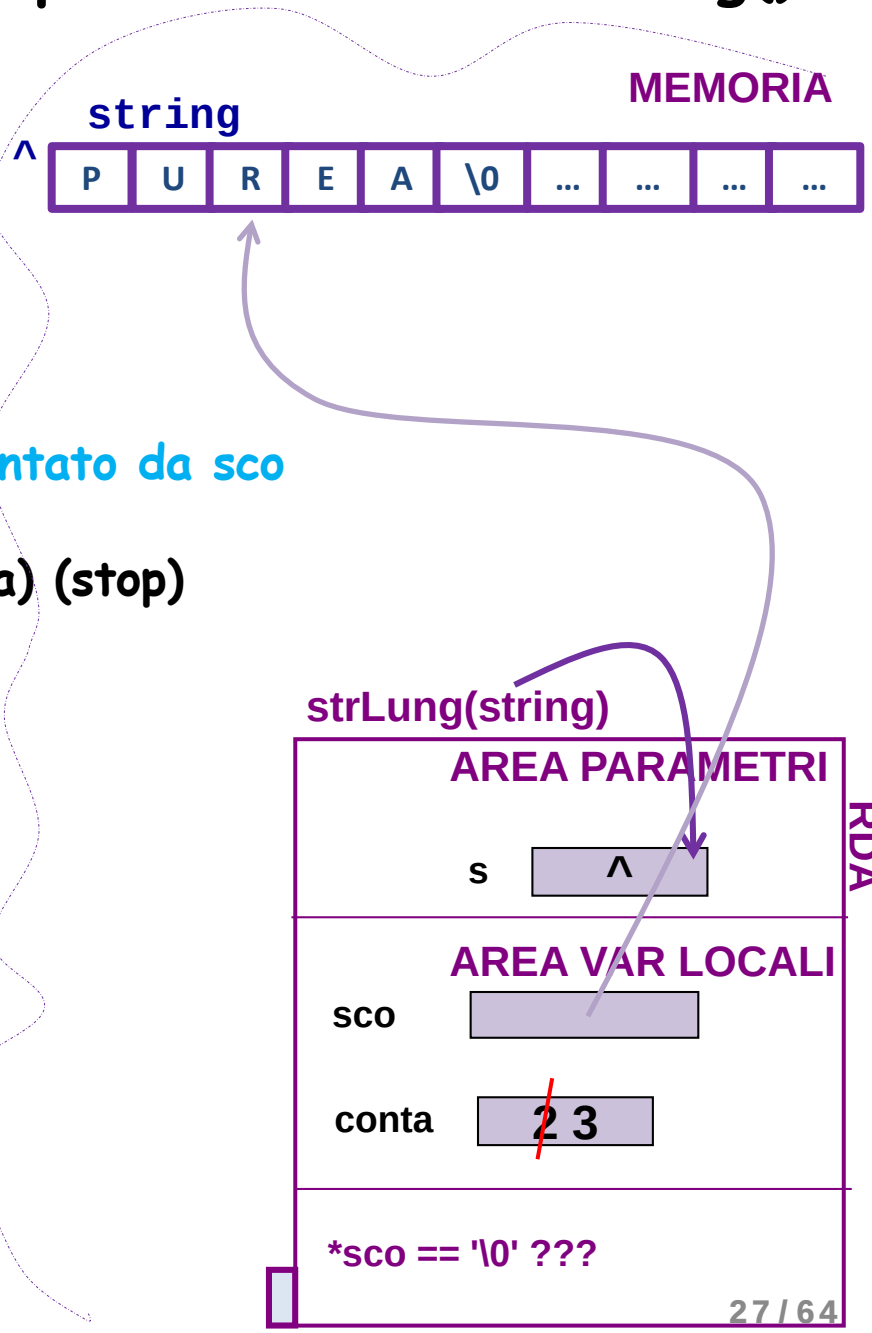
0) int conta, char * sco
(char *s parametro formale)

1) INIT sco = ☺ conta = ☺

secondo passo?
se *sco diverso da '\0' il carattere puntato da sco
e` ☺ quindi … conta ☺
else finita (stringa tutta scandita) (basta) (stop)

2) mentre *sco != '\0'
- conta ☺
- ☺

3) return conta



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

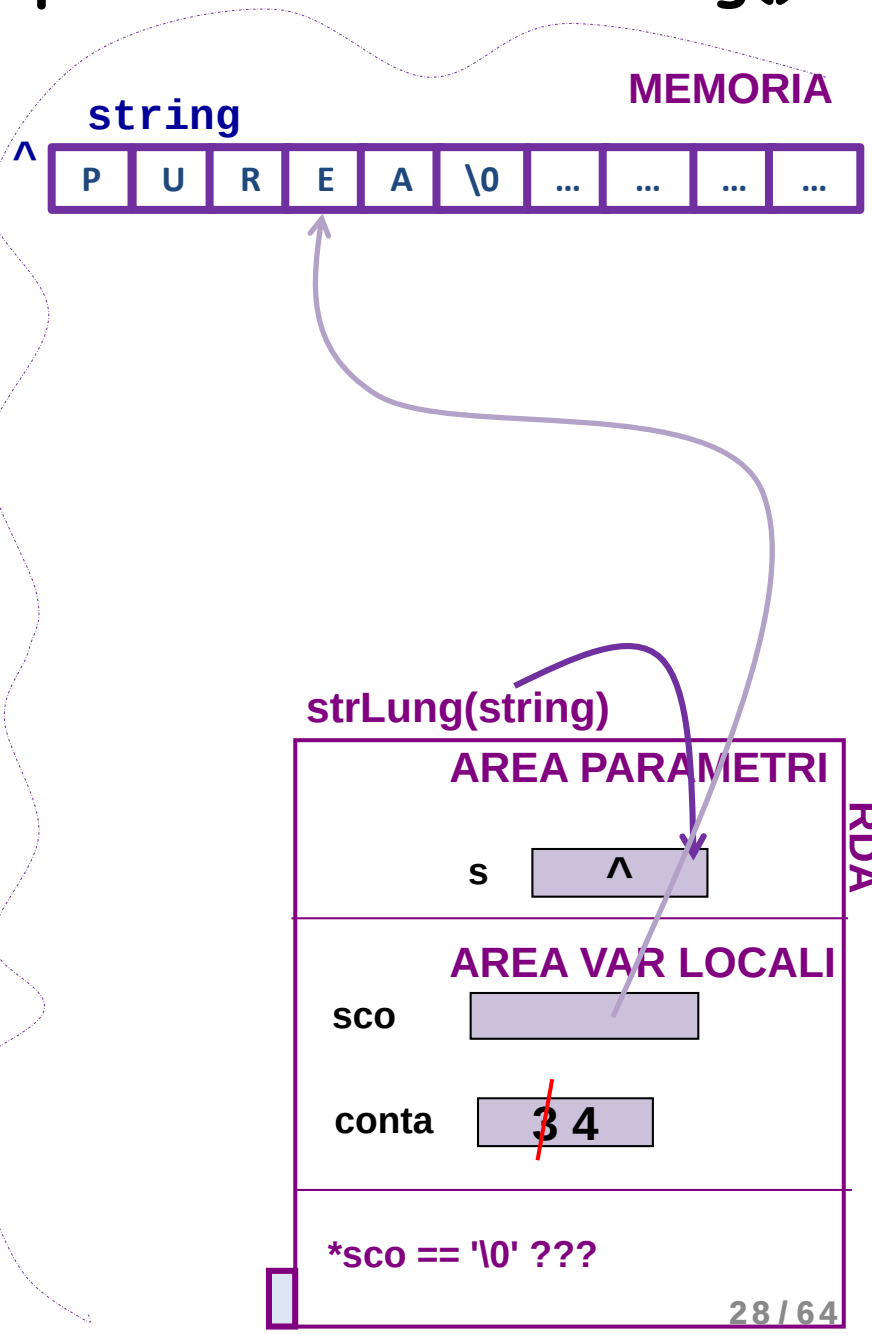
0) int conta, char * sco
(char *s parametro formale)

1) INIT sco = ☺ conta = ☺

secondo passo?
se *sco diverso da '\0'
il carattere puntato da sco
e` diverso da '\0'
quindi e` un carattere significativo
quindi conta deve crescere
else
*sco e` NULL ...
finita (stringa tutta scandita) (basta) (stop)

2) mentre *sco != '\0'
- conta ☺
- ☺

3) return conta



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

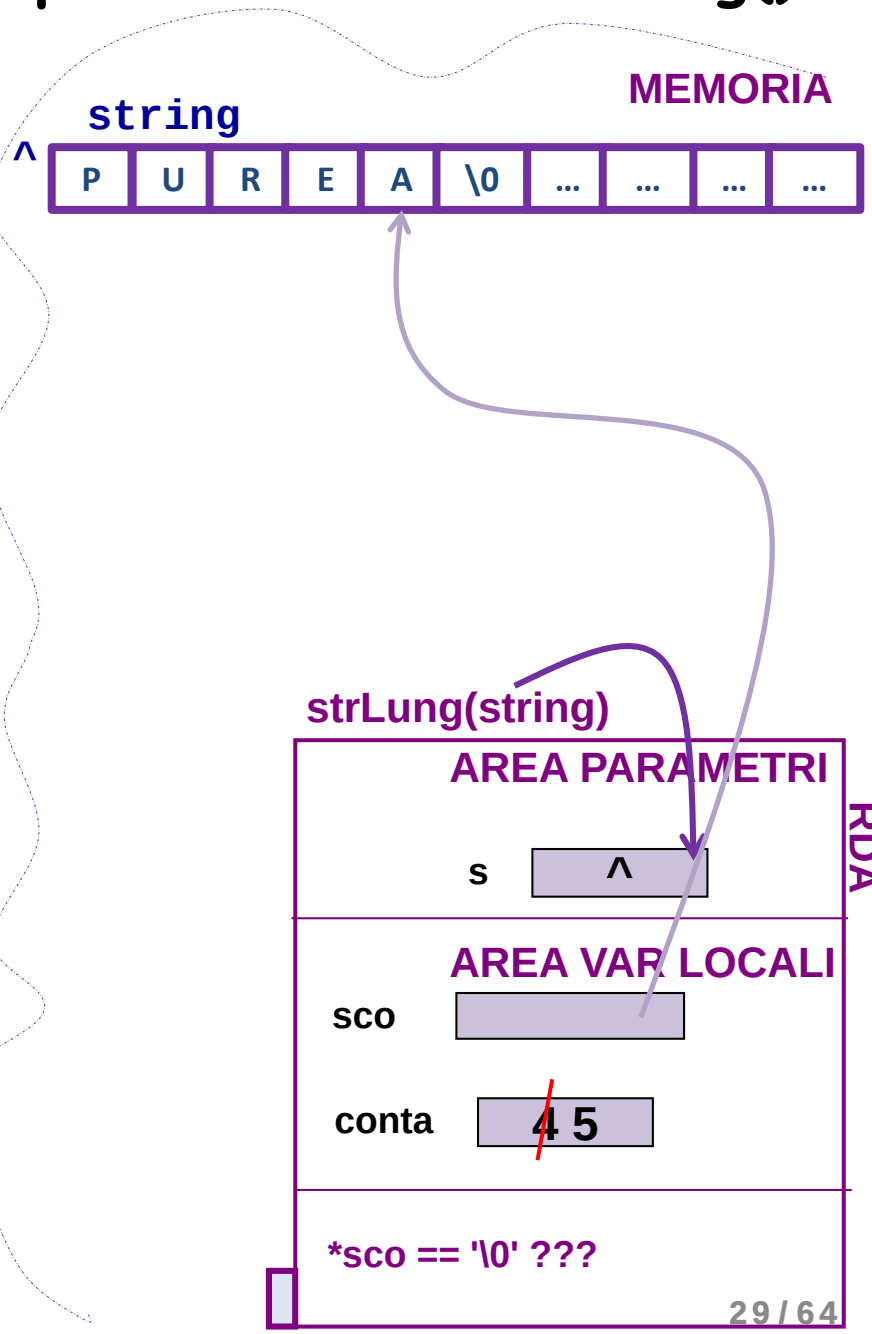
0) int conta, char * sco
(char *s parametro formale)

1) INIT sco = ☺ conta = ☺

secondo passo?
se *sco diverso da '\0'
il carattere puntato da sco
e` diverso da '\0'
quindi e` un carattere significativo
quindi conta deve crescere
else
*sco e` NULL ...
finita (stringa tutta scandita) (basta) (stop)

2) mentre *sco != '\0'
- conta ☺
- ☺

3) return conta



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

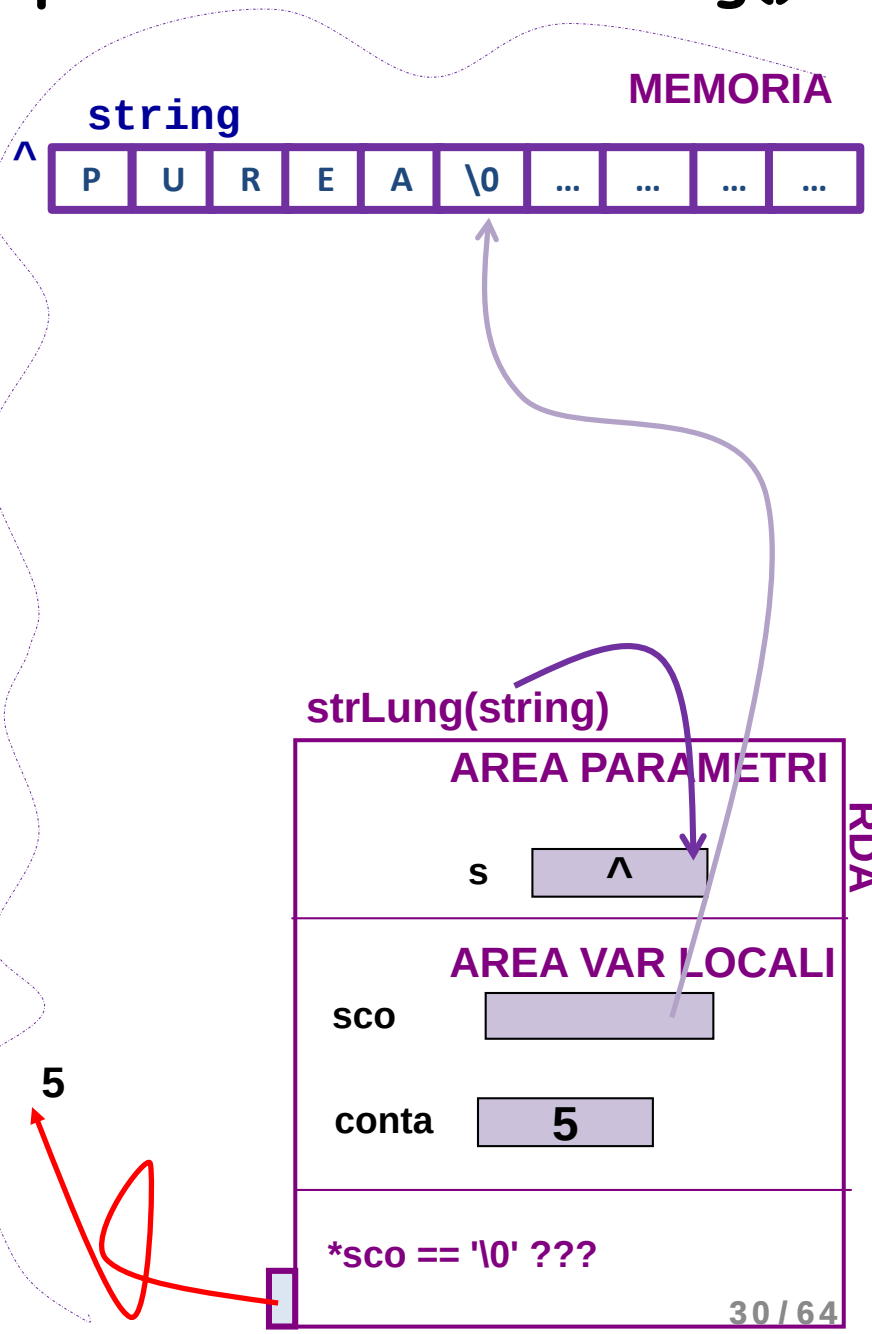
0) int conta, char * sco
(char *s parametro formale)

1) INIT sco = ☺ conta = ☺

secondo passo?
se *sco diverso da '\0'
il carattere puntato da sco
e` diverso da '\0'
quindi e` un carattere significativo
quindi conta deve crescere
else
*sco e` NULL ...
finita (stringa tutta scandita) (basta) (stop)

2) mentre *sco != '\0'
- conta ☺
- ☺

3) return conta



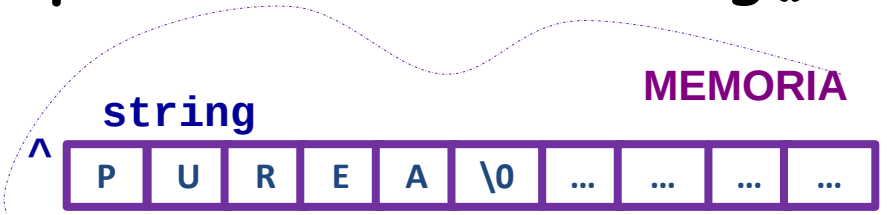
Simulazione di strlen() - Alg. per la funzione strLung()

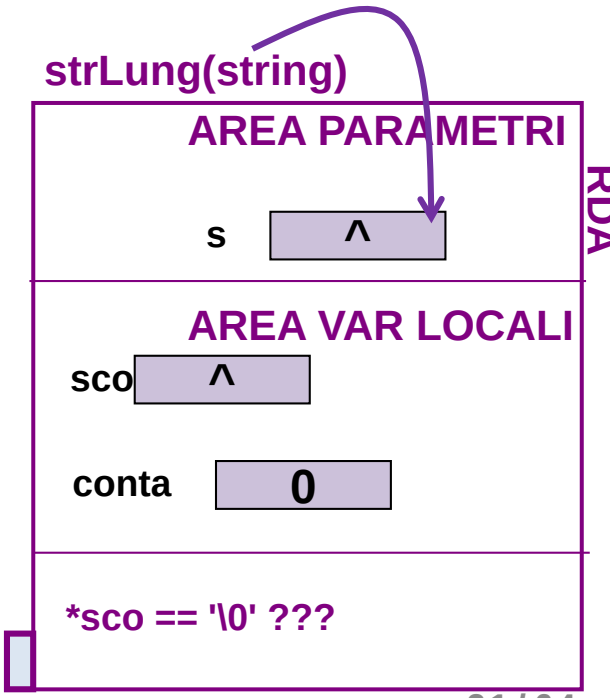
0) int conta, char * sco
(char *s parametro formale)

1) INIT sco = s conta = 0

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

3) return conta



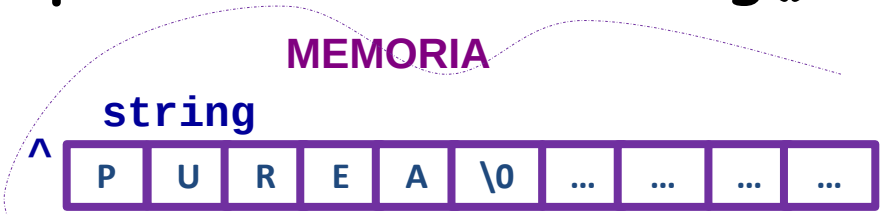


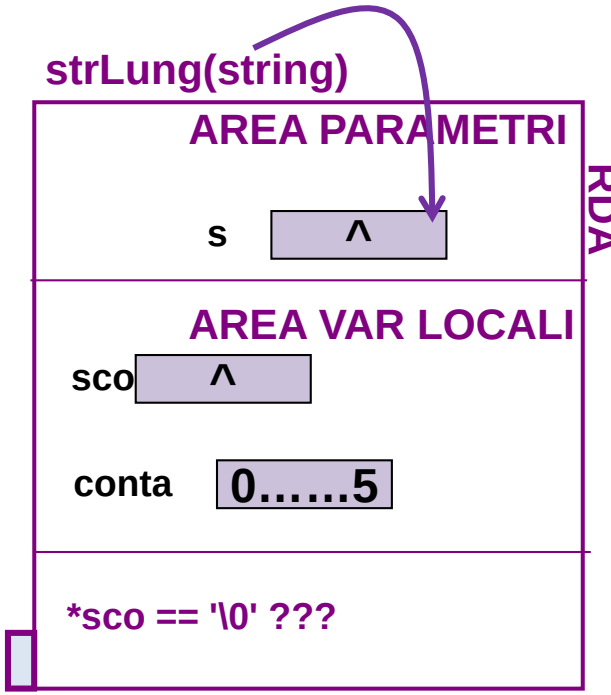
Simulazione di strlen() - Alg. per la funzione strLung()

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

  poi faremo passo 1 e passo 2




return conta;
}
```

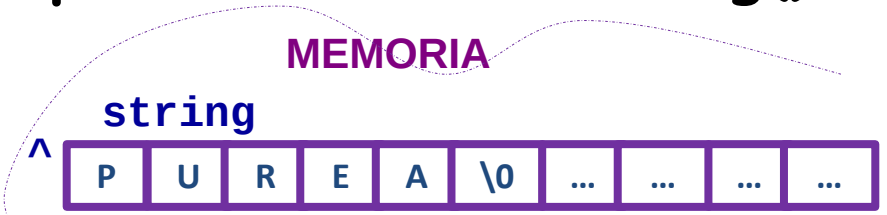


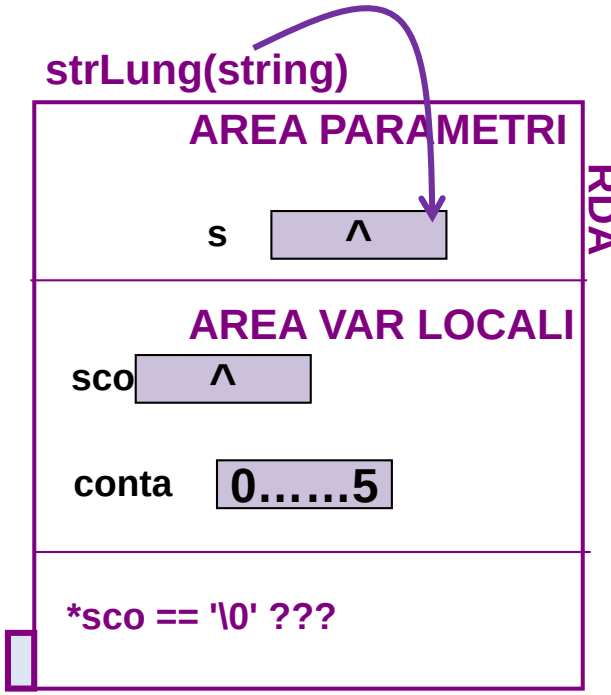


Simulazione di strlen() - Alg. per la funzione strLung()

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;

  ☺ passo 1)

 poi faremo passo 2)



return conta;
}
```

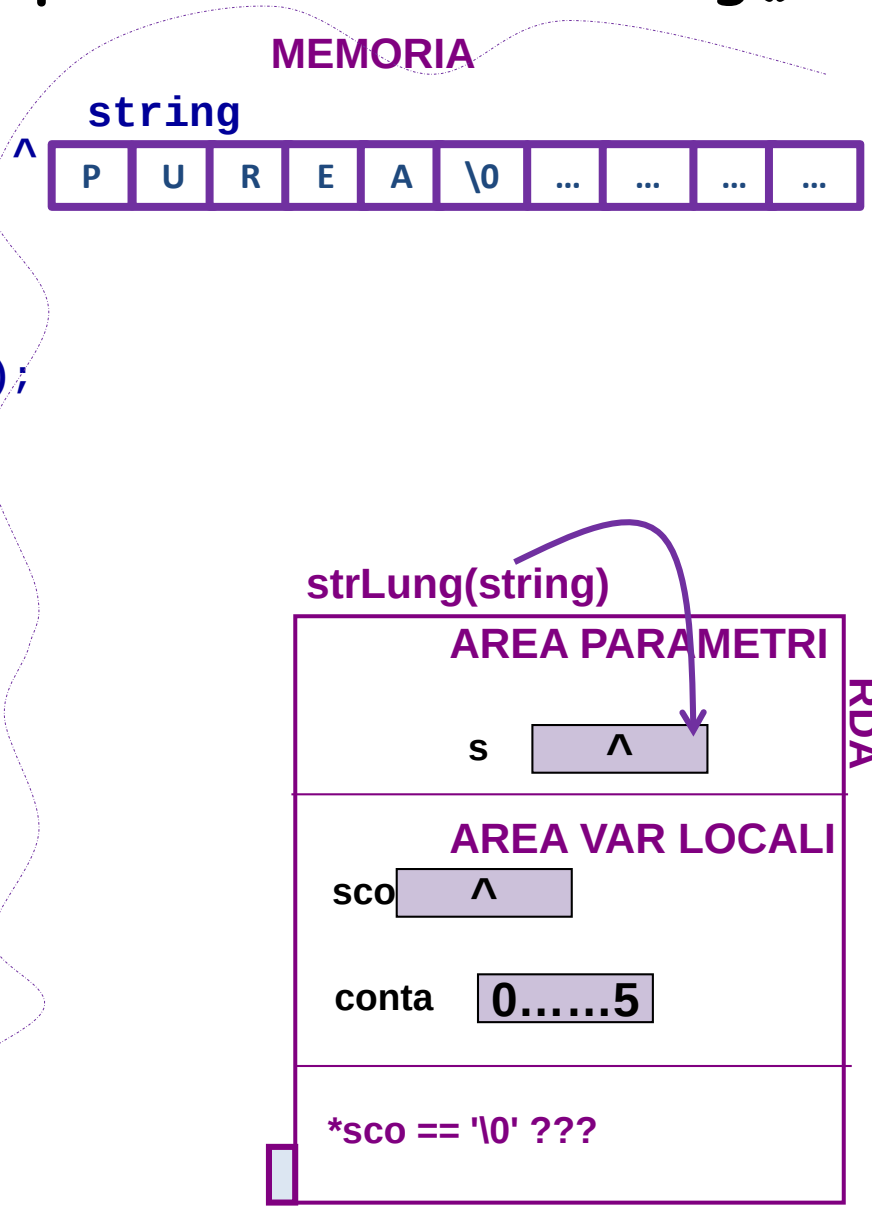




Simulazione di strlen() - Alg. per la funzione strLung()

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;

  ☺ passo 2



return conta;
}
```

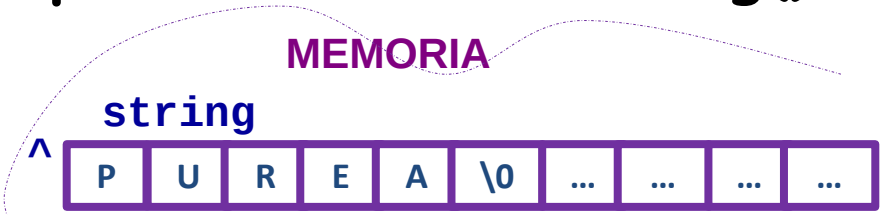


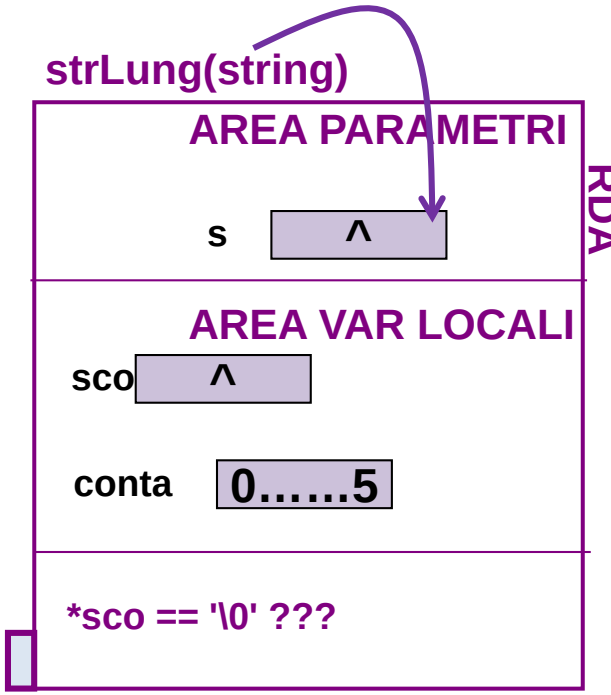
Simulazione di strlen() - Alg. per la funzione strLung()

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;
}
```

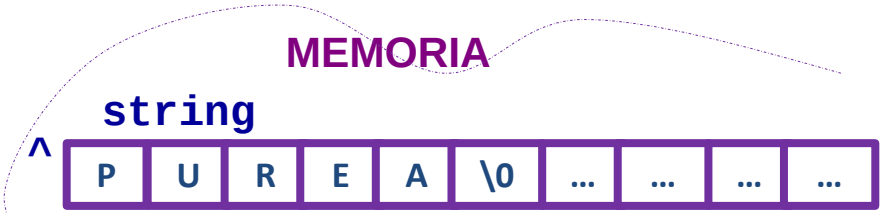
```
int strLung (char *s) {
 char *sco;    int conta;

 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 …

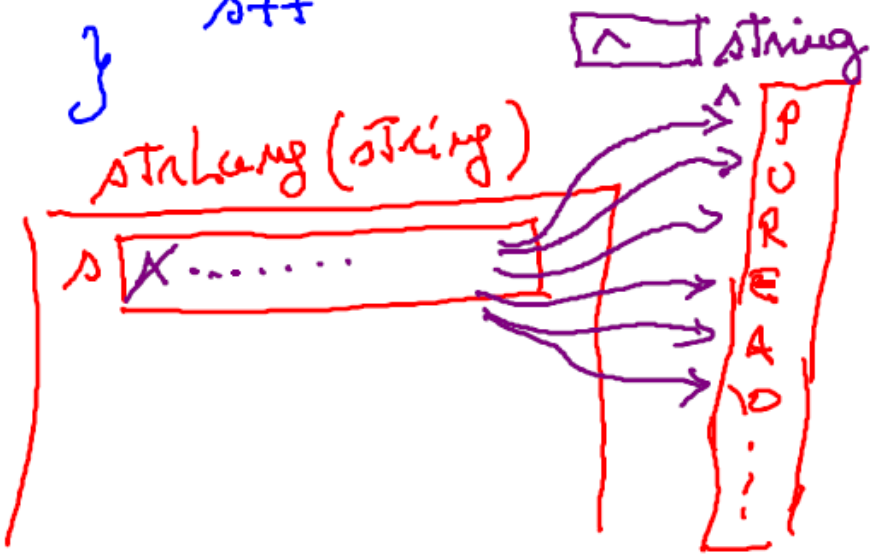
```
int strLung (char *s) {
 char *sco;    int conta;

 sco = s;     conta = 0;
```

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

inoltre, si può anche fare a meno di sco

```
while (*s) {
    conta += 1;
    s++
}
```

QUESTO È VERO PERCHÉ IL PARAMETRO FORMALE s CAMBIA, DURANTE L'ESECUZIONE, SENZA RIFLESSI (EFFETTI COLLATERALI) SUL PARAMETRO ATTUALE string



Variazioni … terza …

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

       conta = 0;
```

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

e ovviamente possiamo usare un FOR invece del WHILE ... se proprio ci piace

```
for ( ; *s ; s++ )
    conta++
```

Variazioni … terza …

o anche

```
for( ; *s ; s++, conta++);
```

COMMA EXPRESSION

```
while (*s) {
  conta++;
  s++;
}
```

comma exp. $exp_1, exp_2, \ldots, exp_n$

VALUTAZIONE

- vengono valutate tutte le espressioni
- una per una, in ordine
- il valore dell'espressione complessiva è quello di exp_n

Esercizi (provateci prima della EG5)

- Simulazione di strcpy()
  - scrivere main(), che chiami la funzione strCopia()
  - e poi void strCopia(char *s1, char *s2)
  - disegnare la mappa di memoria e il RDA per l'esecuzione di strCopia() chiamata dalla main()

- concatenazione di due stringhe
  - scrivere main(), che chiami la funzione concatena(), che riceve tre stringhe e riempie la terza con la concatenazione delle prime due
  - e poi void concatena(char *s1, ...)
  - disegnare la mappa di memoria per l'esecuzione di concatena() chiamata dalla main()

POCA MINESTRA ➔ POCAMINESTRA

Allocazione Statica e dinamica…

ALLOCAZIONE STATICA

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

ALLOCAZIONE DINAMICA

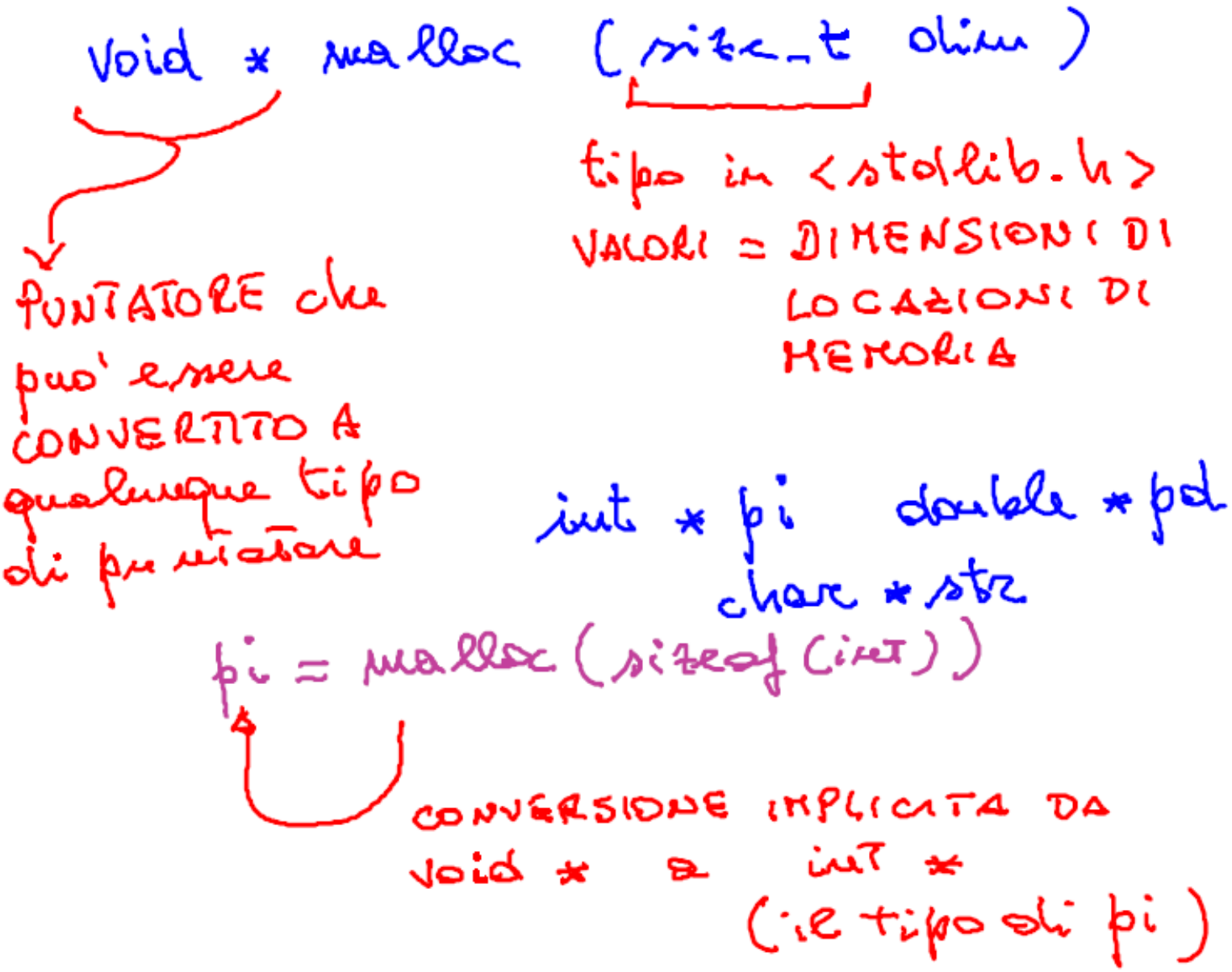
- decisa a run-time
- in conseguenza di una ISTRUZIONE eseguita

↓

`void *malloc (size_t dim)`

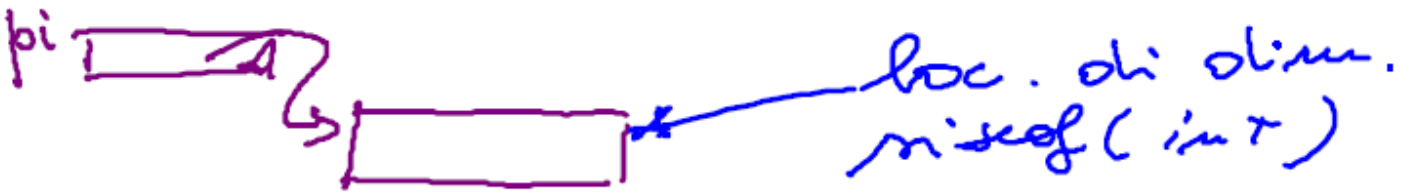
- alloca dim byte
- RESTITUISCE
  - NULL (fallimento)
  - INDIRIZZO DEL BLOCCO DI MEMORIA ALLOCATO

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)

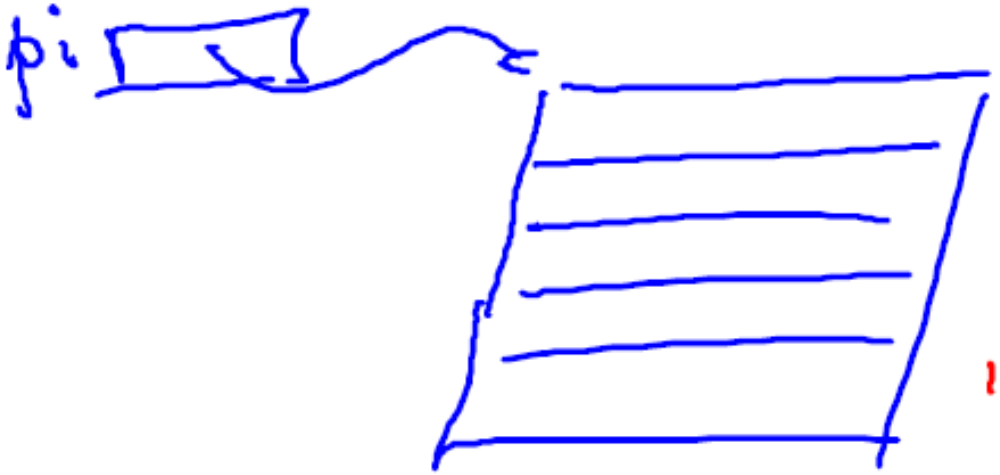
equivalente

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

(sizeof(char) è 1)

Ecco a voi gli array dinamici

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



blocco di 5 ints. interi

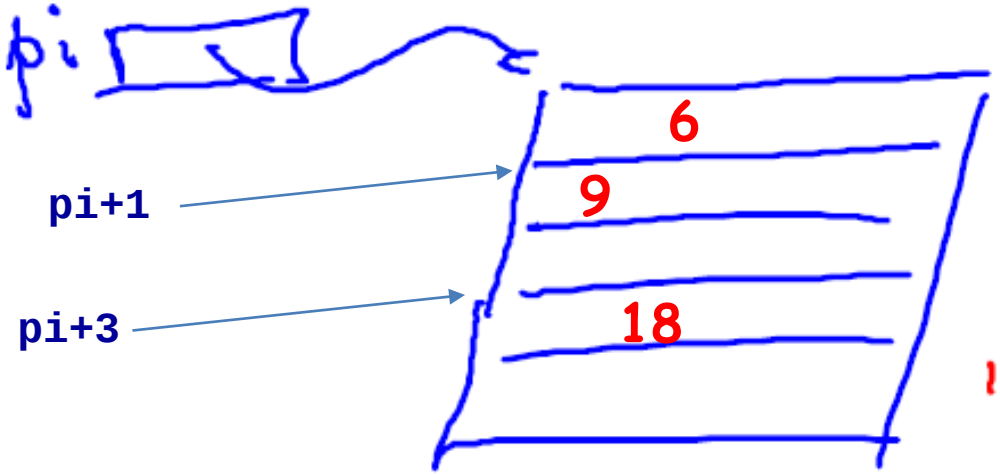
"array dinamico"

che succede se faccio

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

Ecco a voi gli array dinamici

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

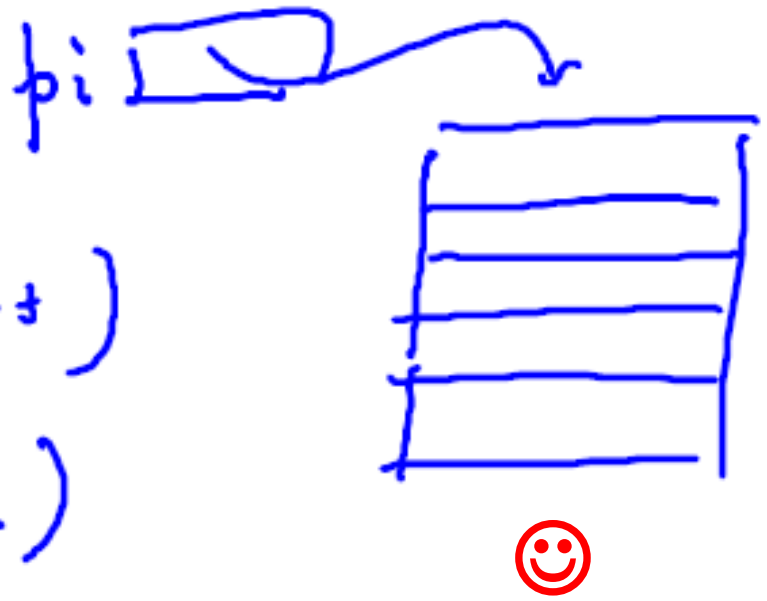


blocco di 5 loc. interi

"array dinamico"

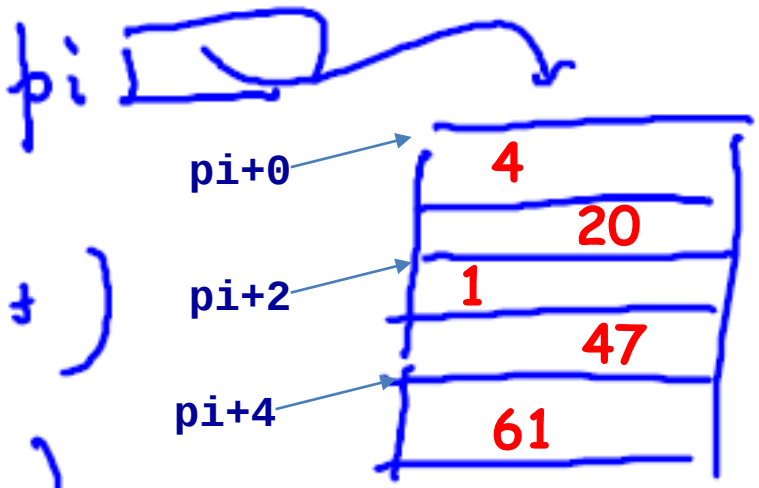
che succede se faccio

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



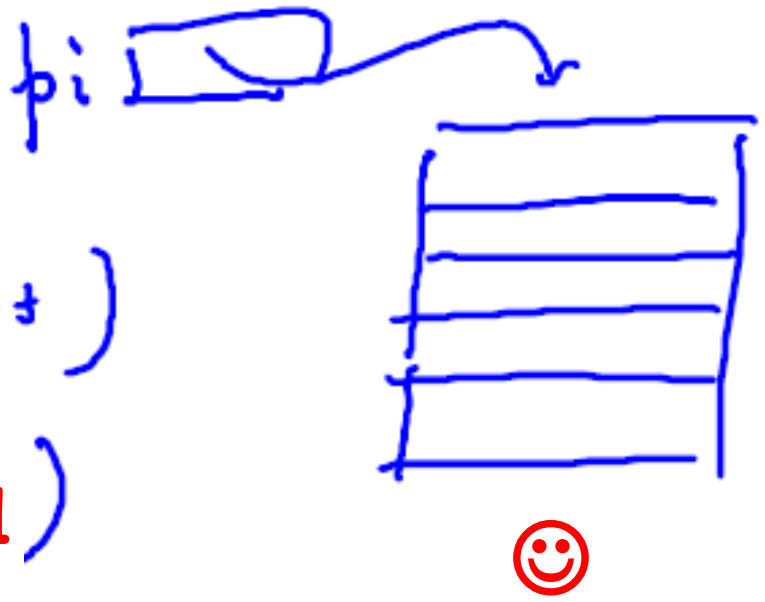

```
for (i=0; i<5; i++)
    scanf("%d", pi+i)
```

INPUT 4, 20, 1, 97, 61



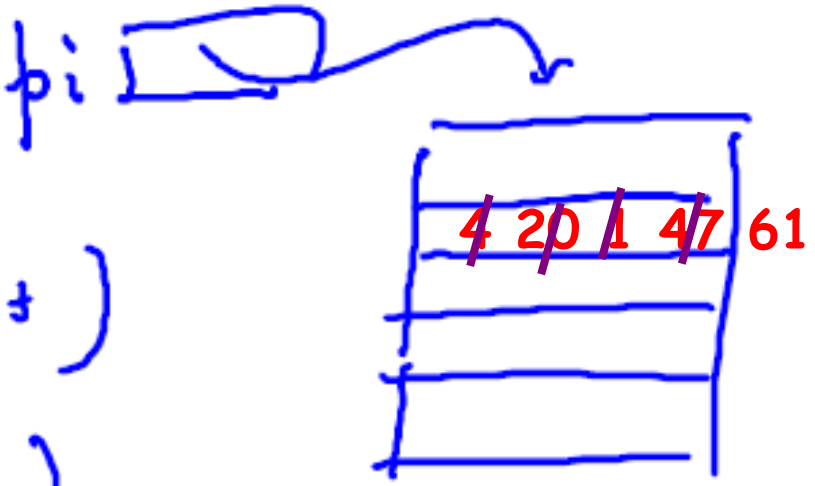

```
for (i=0; i<5; i++)
    scanf("%d", pi+i)
```

INPUT 4 , 20 , 1 , 47 , 61



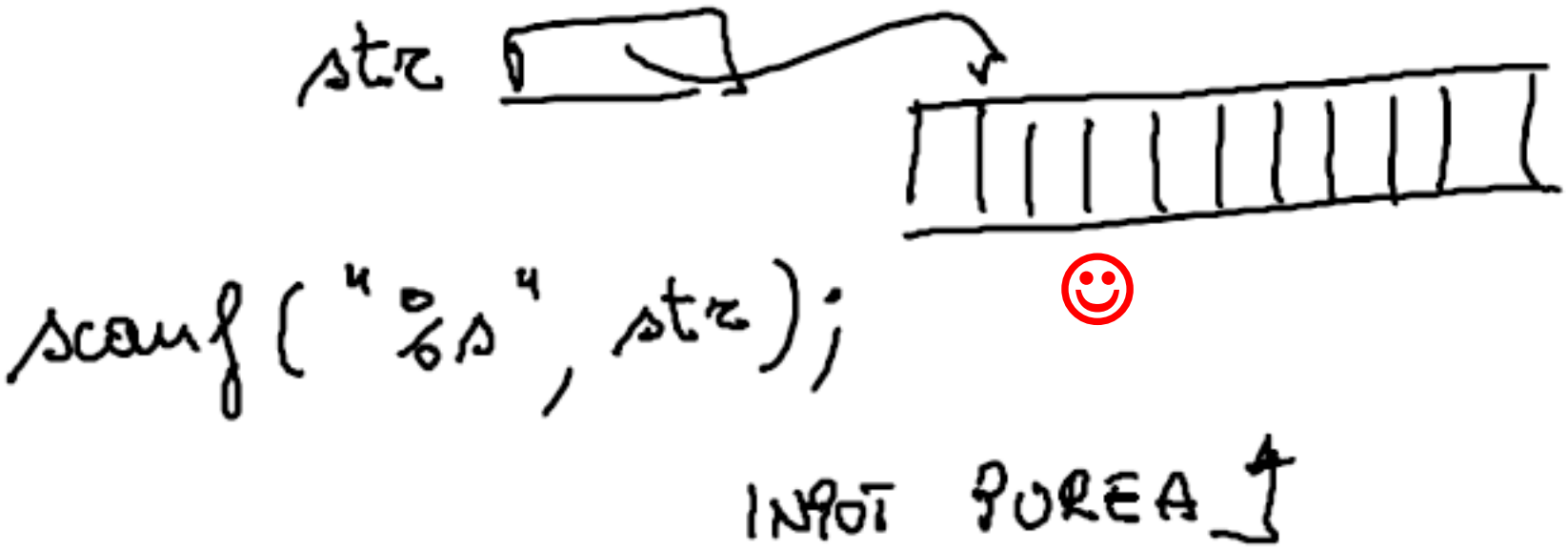

```
for (i=0; i<5; i++)
    scanf("%d", pi+1)
```

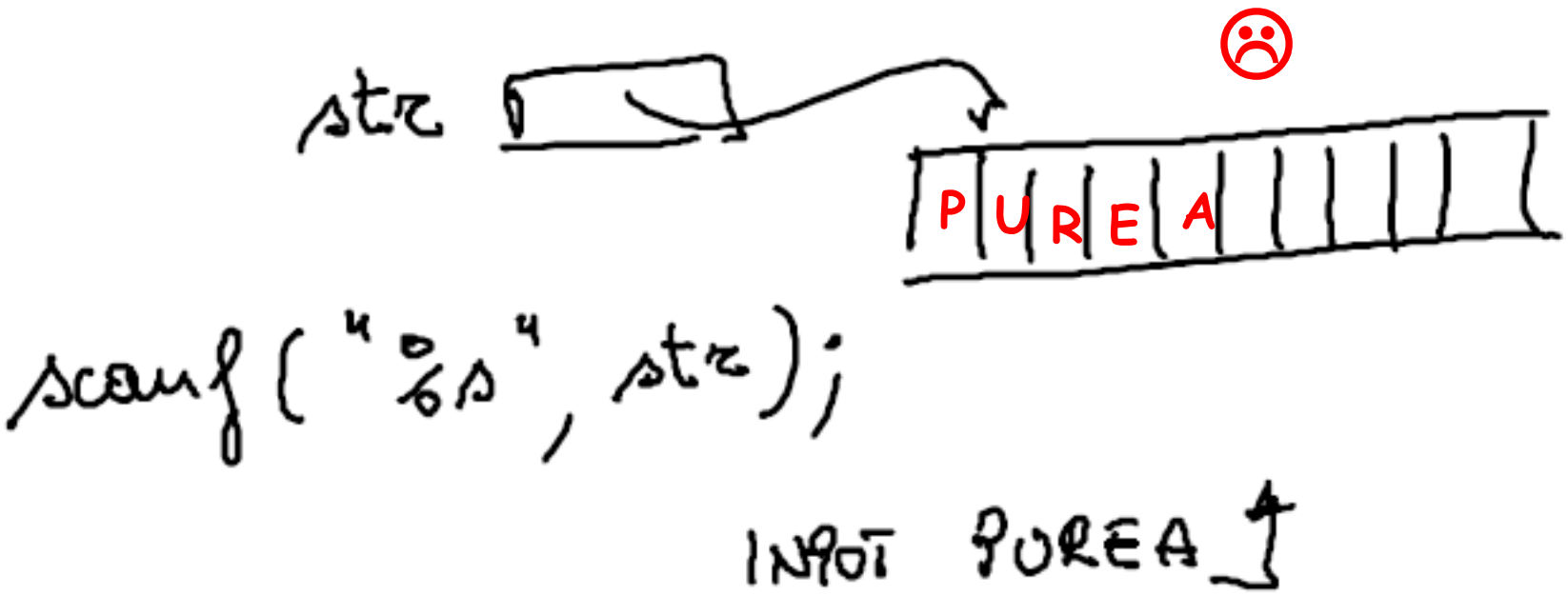
INPUT 4, 20, 1, 97, 61

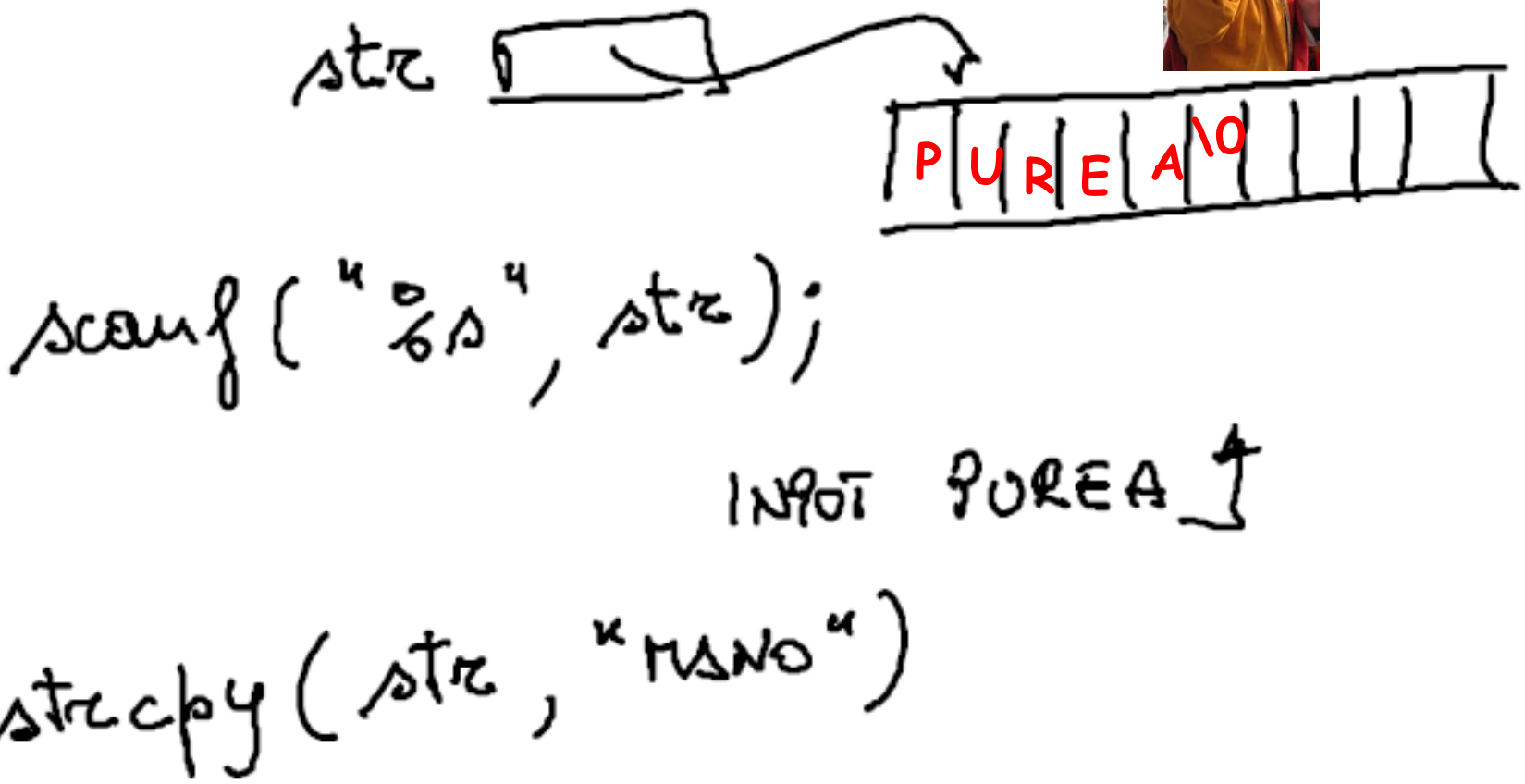


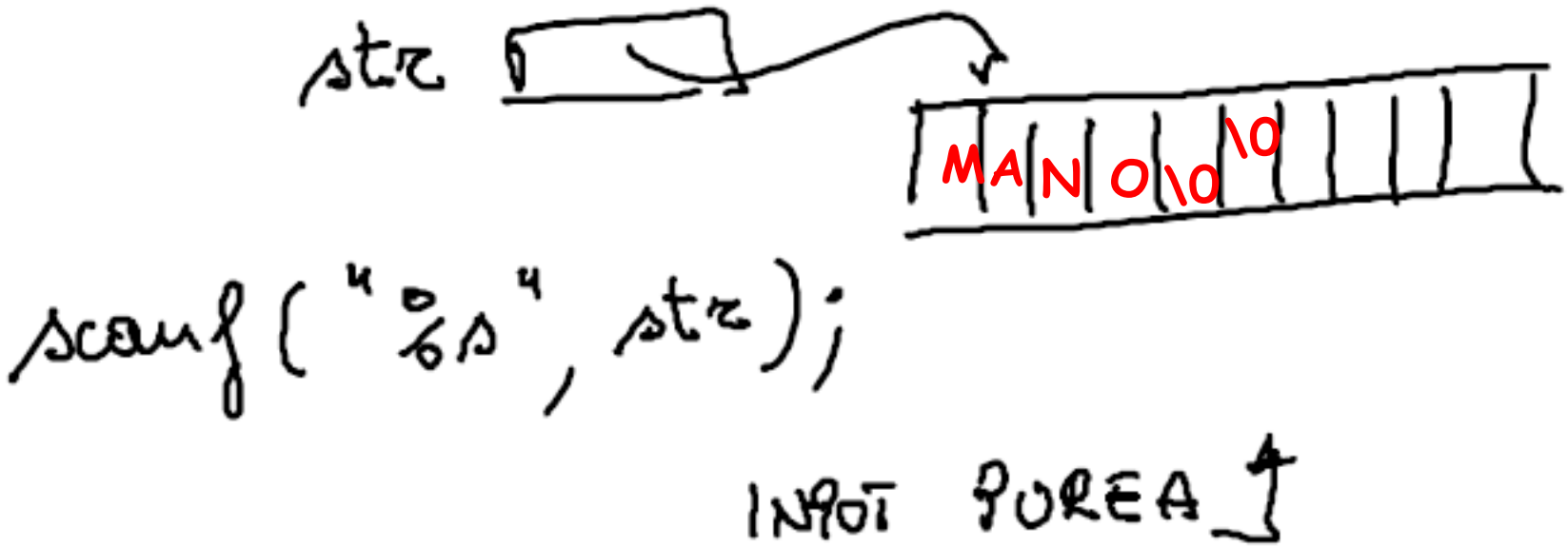

```
for (i=0; i<5; i++)
    scanf("%d", pi+1)
```

INPUT 4 , 20 , 1 , 47 , 61

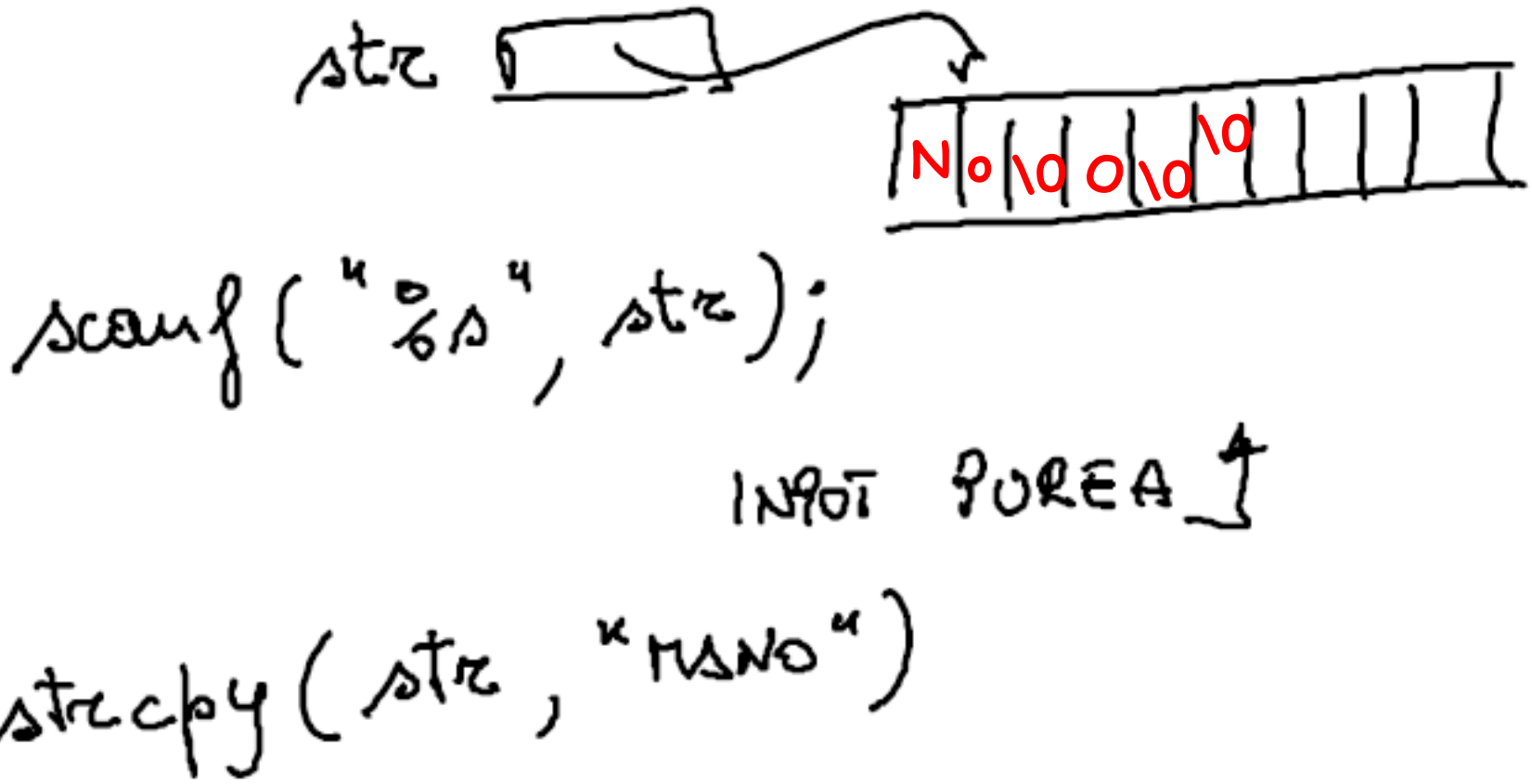

str
scanf("%s", str);
INPUT PUREA


str
P U R E A
scanf("%s", str);
INPUT PUREA


str
P U R E A \0
scanf("%s", str);
INPUT PUREA
strcpy(str, "MANO")




```
strcpy(str, "No");
```




```
strcpy(str, "No");
```

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
```

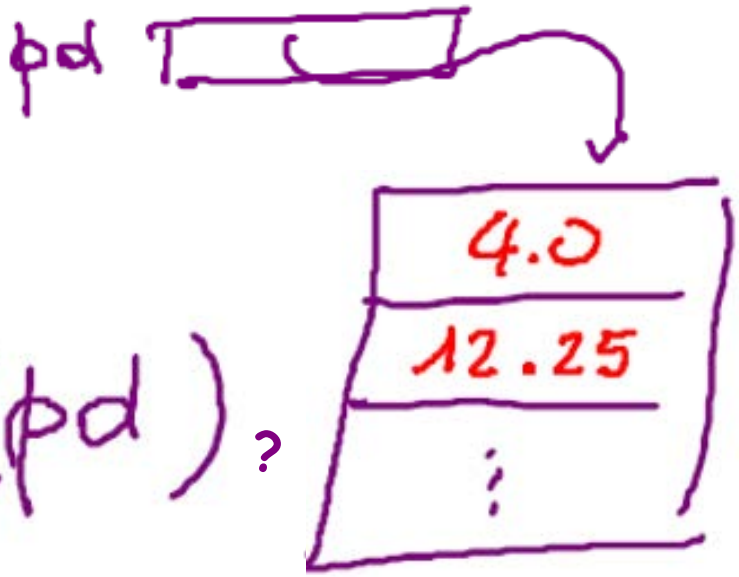
ok, ma se la memoria non serve piu`?

DEALLOCAZIONE

```
void free ( void *punt )
```

LIBERA IL BLOCCO PUNTATO DA punt

- se punt == NULL non succede nulla



che succede se eseguo `free(pd)` ?

ok, ma se la memoria non serve piu`?

DEALLOCAZIONE ----- ma NB

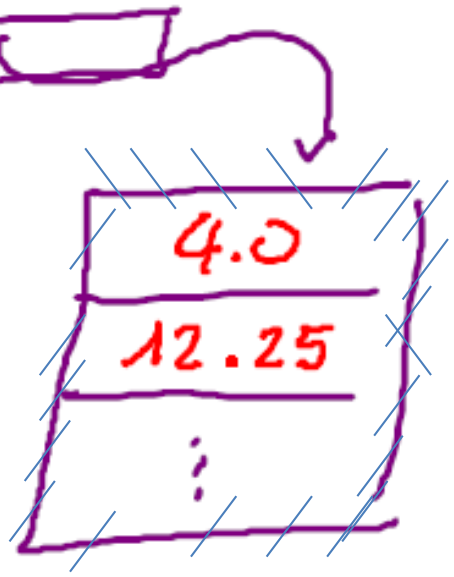
void free (void *punt)

LIBERA IL BLOCCO PUNTATO DA punt

- se punt == NULL non succede nulla

la memoria viene deallocata ma il contenuto non viene cambiato,... e nemmeno pd cambia!

free(pd)



! printf("%g", *(pd+1))

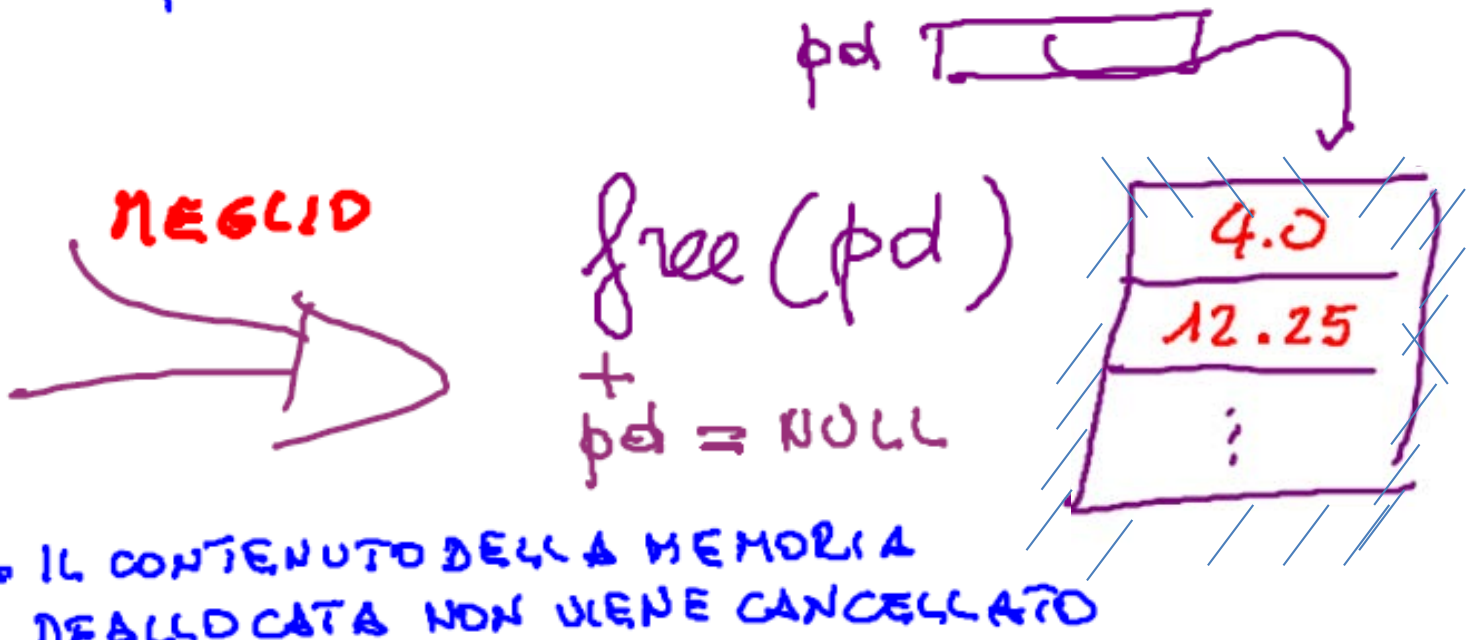
STAMPA 12,25 !

DEALLOCAZIONE --- NB

void free (void *punt)

LIBERA IL BLOCCO PUNTATO DA punt

- il puntatore non cambia contenuto ...
- se punt == NULL non succede nulla



- 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)

==> una volta noto il numero di dati contenuti nel file (lung) si alloca dinamicamente un array di lung double e vi si memorizzano i dati.

SCHEMA DI PROGRAMMA

- algoritmo,
- strutture dati coinvolte,
- main(), con chiamate di funzioni di appoggio (sottoprogrammi per sottoproblemi)
- definizioni delle funzioni (... 1 sola, nel caso inseriamo stampaArray())

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
```

per leggere e contare i dati nel file

```
int lung=0

double dato

double * arrayDouble
```

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) se NON ALLOCATO, messaggio, chiudi file e FINE
      altrimenti
      - rewind file
      - lung letture dal file nel blocco
   d) close file
   e) se il blocco era stato allocato stampalo

4) FINE

e qui useremo una chiamata
stampaArray(arrayDouble, lung)

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

```
int main() {
      char nomefile[15];          FILE *fin;
      double *arrayDouble,        dato;
      int i,                      lung=0;
 printf("nome del file con i double? ");
 scanf("%s", nomefile);
 if ((fin = fopen(nomefile, "r"))==NULL)
     printf("problemi in apertura file.\n");
 else {                  /* file aperto regolarmente */
        while (!feof(fin)) {
             fscanf(fin, "%lf", &dato);
             lung++;
        }              /*NB posizione parentesi*/
 arrayDouble = malloc(lung*sizeof(double));
```

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)
       printf(" problemi in allocazione");
   else {       rewind(fin);
          for(i=0; i<lung; i++)
              fscanf(fin, "%lf", &arrayDouble[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 ... */
     for (i=0; i<lung; i++)
          printf(" %g\n", arrayDouble[i]);
}      /* 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

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

```
  arrayDouble = malloc(lung*sizeof(double));
   if (arrayDouble==NULL)
     printf(" problemi in allocazione"); /* e chiusura file !! */
   else {       rewind(fin);
          for(i=0; i<lung; i++)
                                  equivalente a arrayDouble+i
              fscanf(fin, "%lf", &arrayDouble[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 ... */
    for (i=0; i<lung; i++)
                                  equivalente a *(arrayDouble+i)
          printf(" %g\n", arrayDouble[i]);
}      /* 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

Stampaarray??? (esercizio: def. Funzione e inserire chiamata al posto giusto nel programma ...)

4) FINE