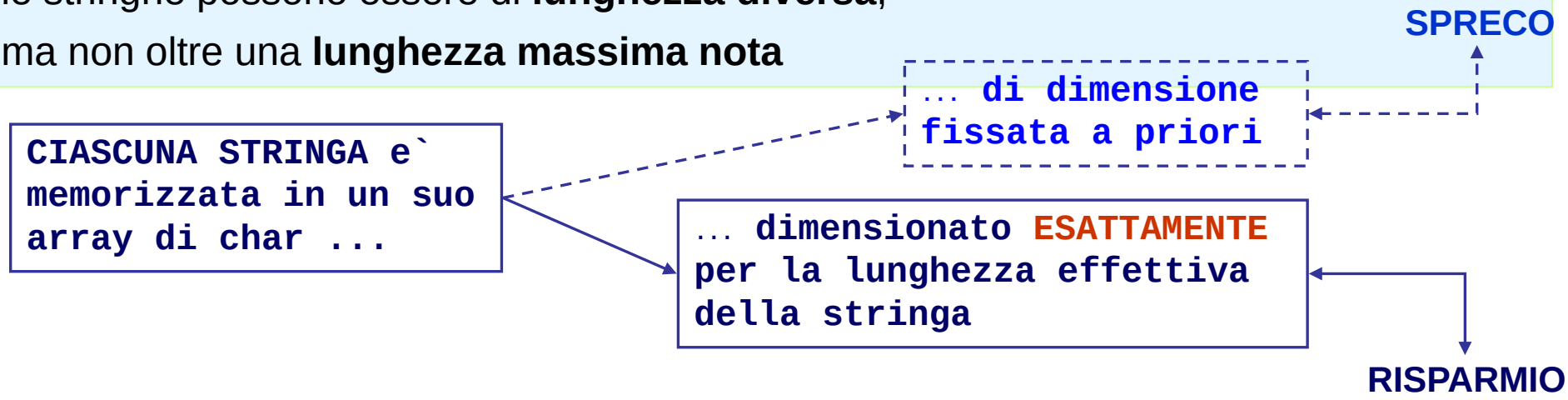
Tecniche della Programmazione, lez.16

Uso dell'allocazione dinamica; gestione di stringhe; gestione di una struttura dati per una collezione di stringhe

- allocazione dinamica di (tante) stringhe ("esatte") in un programma
- array di stringhe ("esatte"): operazioni di "aggiunta" e "ricerca"
- programma di gestione stringhe
- struttura dati piu` complessa per una collezione di stringhe
- funzionalita` classiche

Allocazione Dinamica: Stringhe Esatte

Problema **gestione di MOLTE STRINGHE** alfanumeriche nel programma;
le stringhe possono essere di **lunghezza diversa**,
ma non oltre una **lunghezza massima nota**



SCHEMA DI REALIZZAZIONE

- **vengono definite le diverse stringhe da usare nel programma, come puntatori;**

```
char * str, *str2, *str3, *str4;  /* 4 stringhe (in potenza) */
```

- **viene definita una "stringa buffer" abbastanza grande per contenere qualunque stringa da gestire;**

```
char buffer[LUNGMAX+1]
```

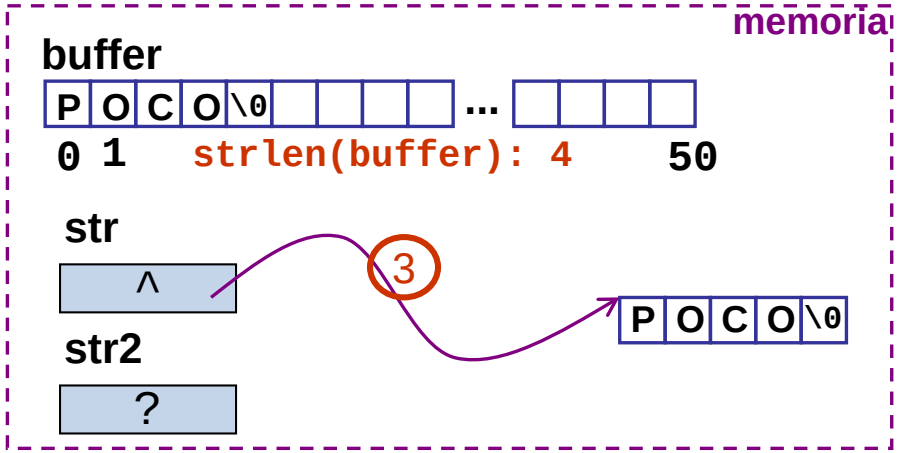
- **per ogni stringa da memorizzare, prima la si legge usando buffer e poi la si alloca+riempie/assegna in corrispondenza di uno dei puntatori, in modo che occupi solo la memoria necessaria**

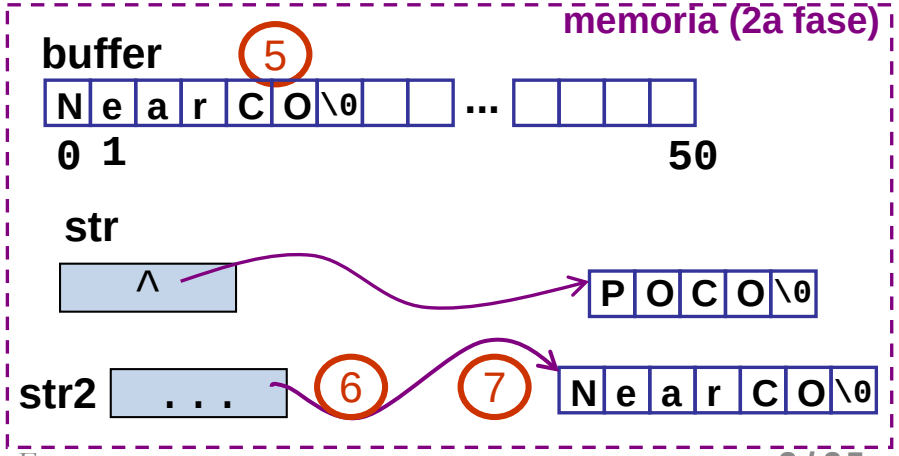
...

Allocazione Dinamica: Stringhe Esatte

Problema gestione di MOLTE STRINGHE ...

```
#include <stdio.h>
#define LUNGMAX 50    /* stringhe mai piu` lunghe di 50 */
...
1 char buffer[LUNGMAX+1], *str, *str2 ...
...
2 scanf(...%s...", buffer);

3 str=malloc(strlen(buffer)+1);
  if (str)
     strcpy(str, buffer); 4
  else ... /* messaggio di errore*/
...
5 scanf(...%s...", buffer);

6 str2=malloc(strlen(buffer)+1);
  if (str2)
     strcpy(str2, buffer); 7
  else ...
```

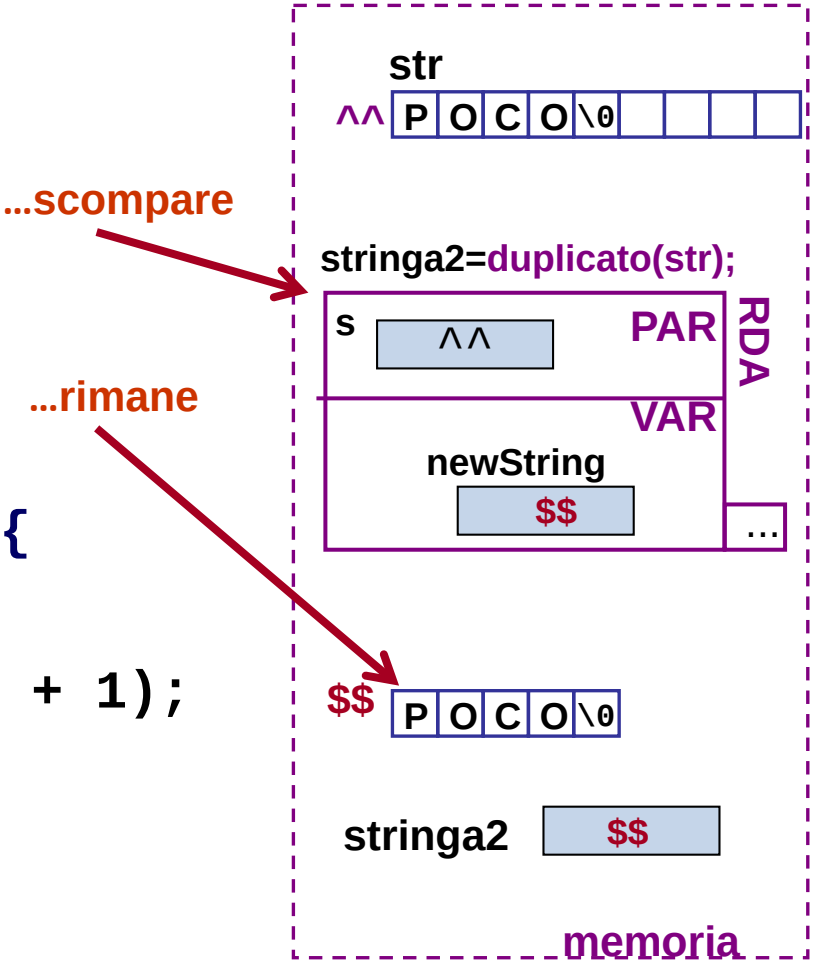




duplicazione (esatta) di una stringa

esercizio funzione che
ricevendo una stringa **s** **restituisca** una copia (esatta) di s

```
#include <stdio.h>
#include <stdlib.h>
... (dich.) ...
int main() {
  char str[9],    *stringa2;
.../* "POCO" in stringa2 */
 stringa2 = duplicato(str);
...
return 0;
}

      char * duplicato (char *s) {
        char * newString;

       newString=malloc(strlen(s) + 1);
       if(newString)
         strcpy(newString, s);

      return newString;
      }
```

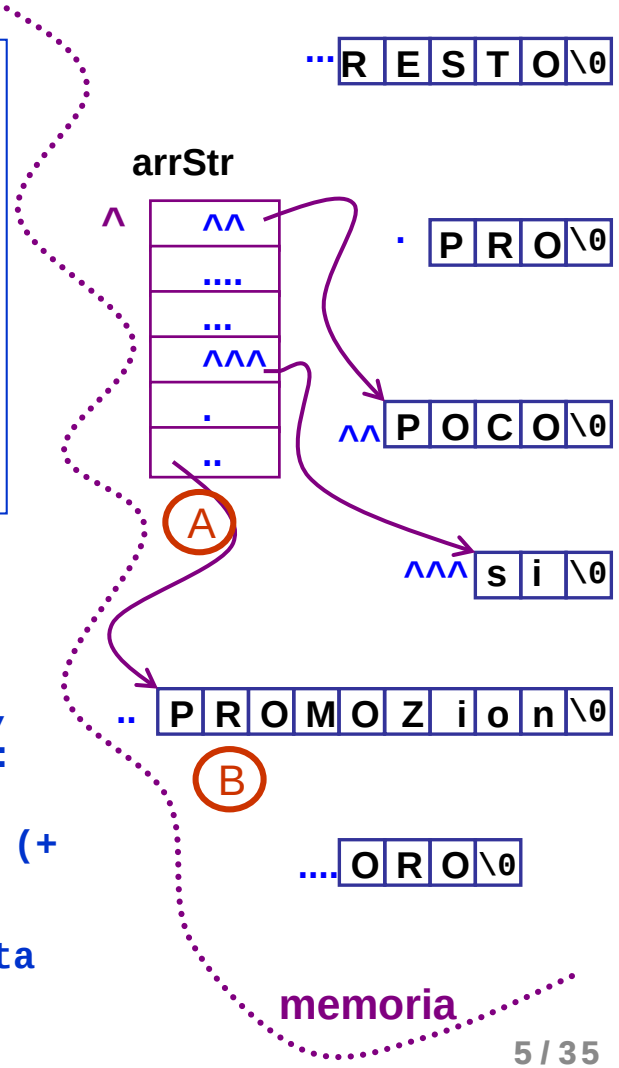


Gestione di tante stringhe: Array di stringhe

Array di stringhe
`char * arrStr[6];`

array di puntatori;
ogni elemento punta ad una stringa
(una stringa e` un blocco/array di caratteri)

```
arrStr[5] = malloc(10);      A  /* allocazione della memoria esattamente
                                necessaria per una delle
                                stringhe (9 char + il '\0') */

if (arrStr[5] == NULL)
    printf("ERRORE IN ALLOCAZIONE MEMORIA\n");
else                         B  /* la memoria disponibile viene
                                   riempita esattamente  */
    strcpy(arrStr[5], "PROMOZion");
```



NB - arrStr[6] non e` una locazione dell'array

- qualunque arrStr[i](i=0...5) e` un puntatore;

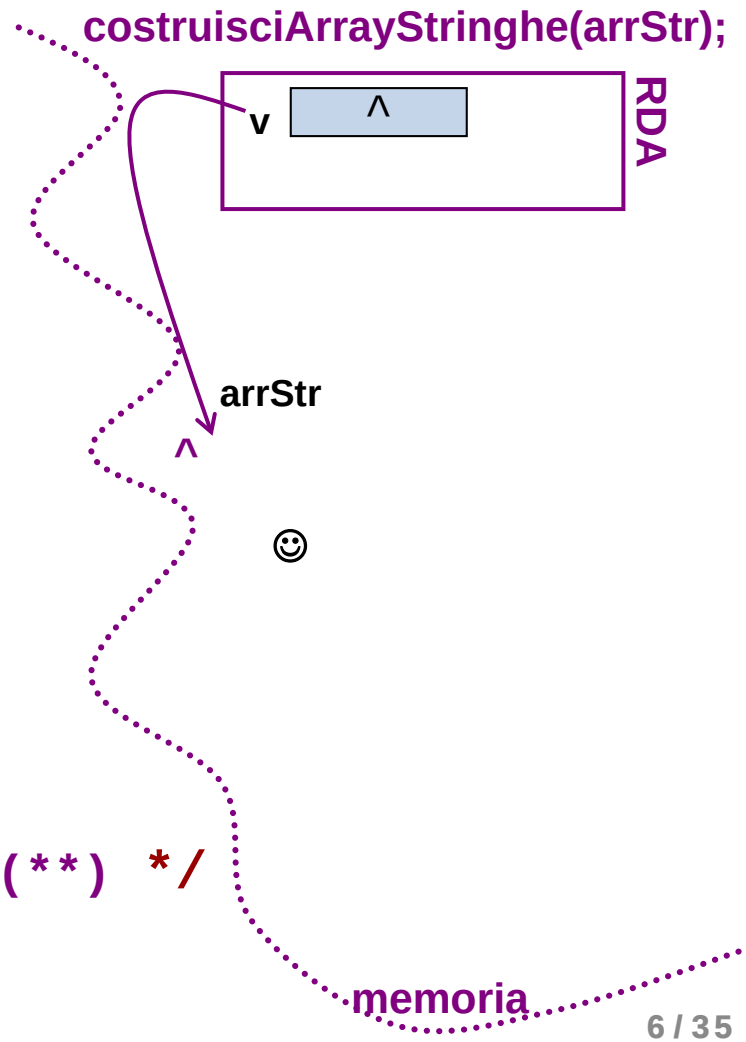
- quando arrStr[5] punta ad un blocco di (9+1) caratteri, arrStr[5] e` l'indirizzo iniziale di un array di 10 char: passando questo indirizzo a strcpy, si puo` copiare nell'array puntato una stringa di al massimo 9 caratteri (+ un carattere di fine stringa, '\0');

- ovviamente, questo array lo abbiamo dimensionato apposta per (9+1) caratteri e lo usiamo tutto!

Array di stringhe (lettura) - 1/3 -

esercizio funzione costruisciArrayStringhe che
ricevendo un array di stringhe, **char * v[N]**,
legga **N stringhe**, ciascuna di al piu` 80 char, **e le memorizzi nell'array** **(esatte)**

```
/* 1a fase: ambiente di calcolo */
```

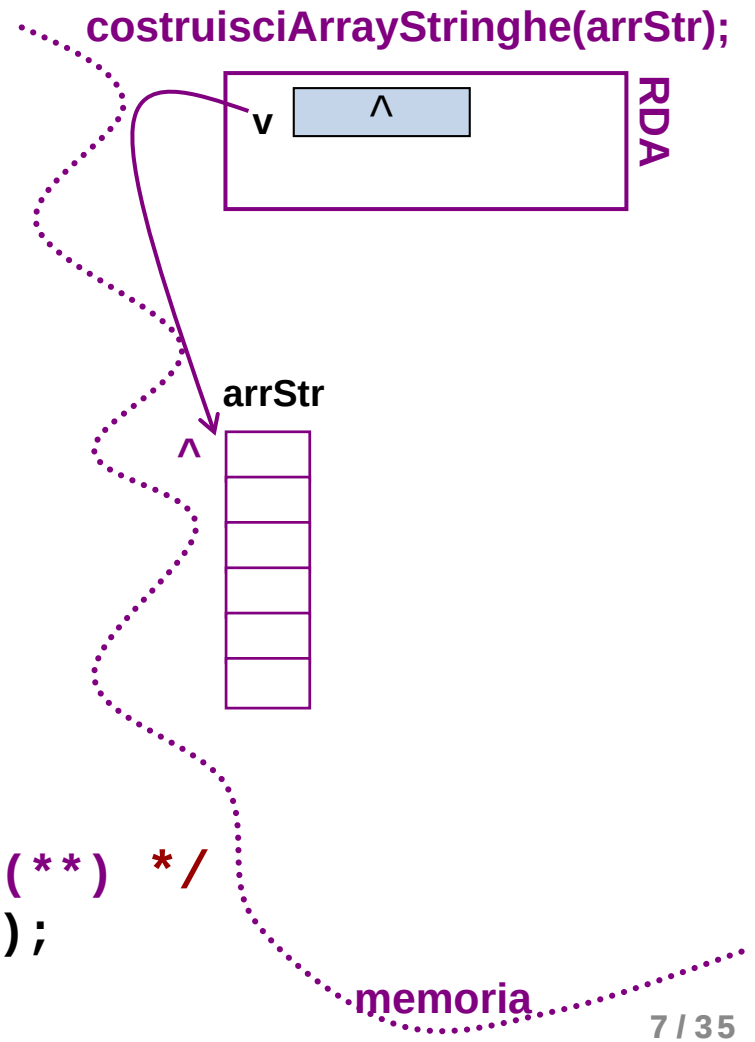



```
/* 2a fase: PROTOTIPO (dichiarazione) (**) */
void costr....
```

Array di stringhe (lettura) - 1/3 -

esercizio funzione che
ricevendo un array di stringhe, **char * v[N]**,
legga **N stringhe**, ciascuna di al piu` 80 char, **e le memorizzi nell'array** **(esatte)**

```
/* 1a fase: ambiente di calcolo */
#include <stdio.h>
#include <stdlib.h>
#define N 6
#define LUNGMAX 80
... (**) ...
int main() {
  char * arrStr[N];
...
 costruisciArrayStringhe (arrStr);
...
return 0;
}

/* 2a fase: PROTOTIPO (dichiarazione) (**) */
void costruisciArrayStringhe (char * []);
```

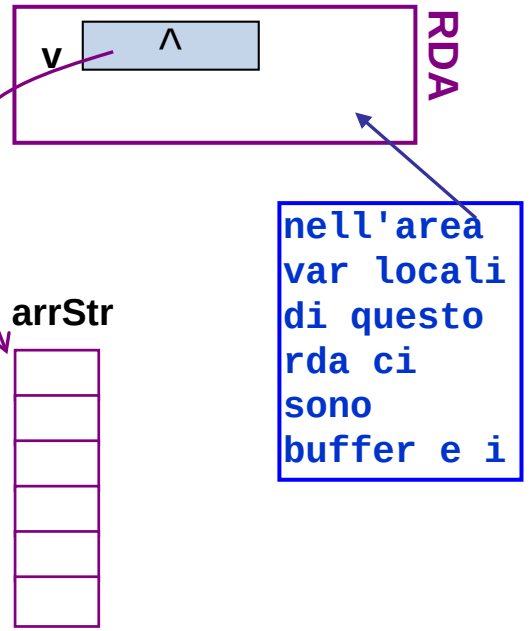


Array di stringhe (lettura) - 2/3 -

continua funzione che legge un array di N stringhe, ciascuna di al piu` 80 char

```
/* 3a fase: definizione funzione */
void costruisciArrayStringhe(  ☺          );
  char buffer[LUNGMAX+1];
  int i;

 for (i=0; i<N; i++) {

 } /* fine for */

return;
}
```

costruisciArrayStringhe(arrStr);



si tratta di leggere una sequenza di stringhe date in input (POCO, ORO, RESTO, Si`, PRO, PROMOZion), memorizzandole secondo l'ordine di input in arrSTr:

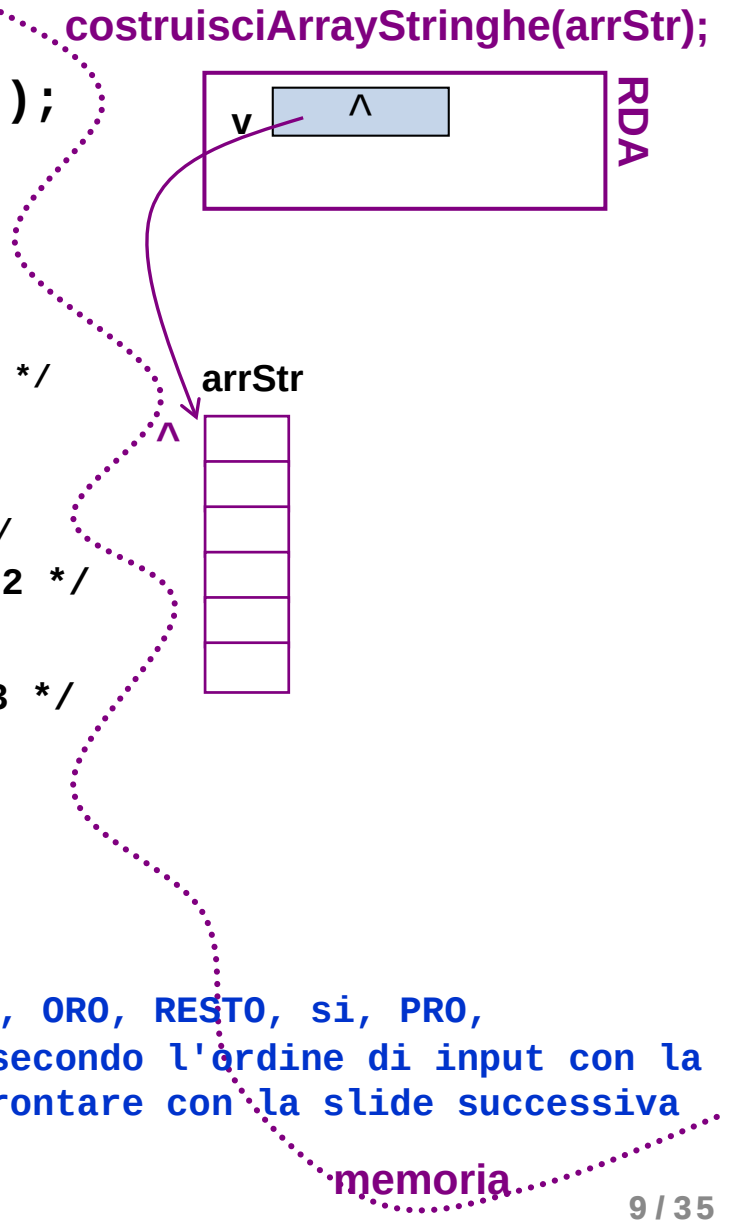
1) iterare
   - 1.1) leggere stringa in buffer
   - 1.2) allocare stringa i-esima in arrStr
   - 1.3) copiare da buffer in i-sima ...

memoria

Array di stringhe (lettura) - 2/3 -

continua funzione che legge un array di N stringhe, ciascuna di al piu` 80 char

```
/* 3a fase: definizione funzione */
void costruisciArrayStringhe(char * v[N]);
  char buffer[LUNGMAX+1];
  int i;

 for (i=0; i<N; i++) {
                        /* lettura di una stringa ... */
   printf("scrivi una str ...\n");
   scanf("%s", buffer);
                        /* ... e sua memorizzazione */
   v[i] = malloc(strlen(buffer)+1); /* 1.2 */
   if (v[i])
      strcpy(v[i], buffer);          /* 1.3 */
   else {
      printf("eeekkk\n");
      break;
   }
 } /* fine for */

return;
}
```



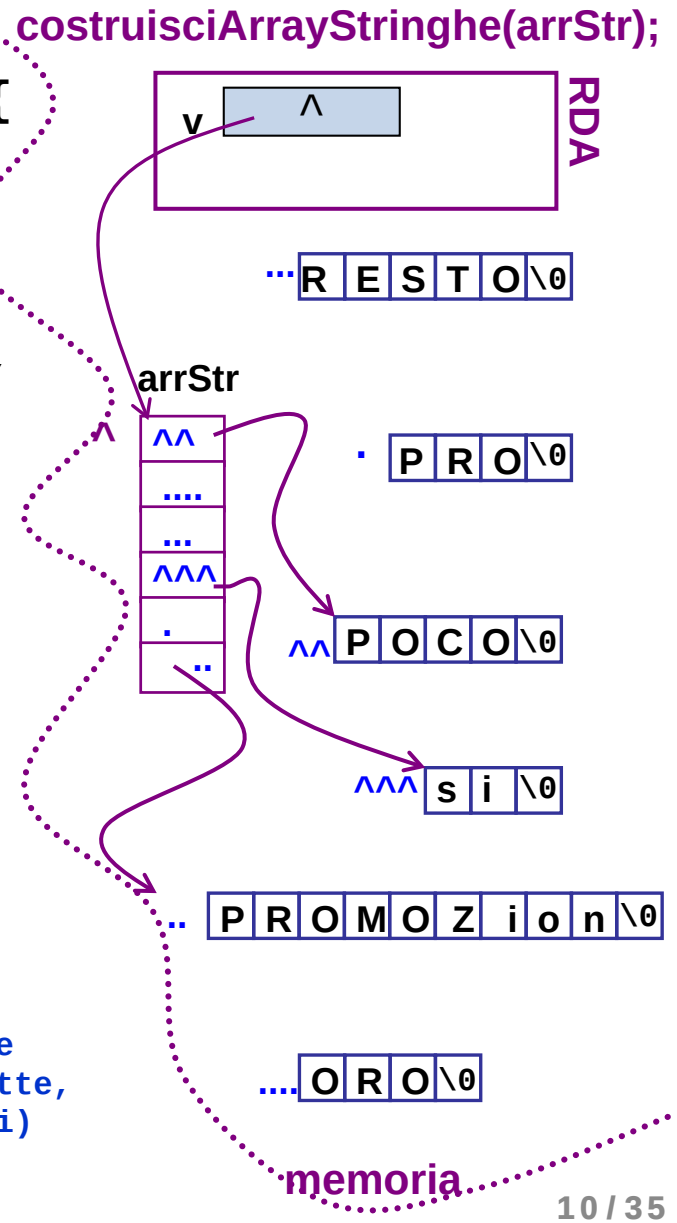
☺ leggere le stringhe POCO, ORO, RESTO, si, PRO, PROMOZion, memorizzandole secondo l'ordine di input con la funzione a lato, e poi confrontare con la slide successiva

Array di stringhe (lettura) - 3/3 -

funzione che legge un array di N stringhe, ciascuna di al piu` 80 char

```
/* 3a fase: definizione funzione */
void costruisciArrayStringhe(char * v[N]){
  char buffer[LUNGMAX+1];
  int i;

 for (i=0; i<N; i++) {
                          /* lettura di una stringa ... */
   printf("scrivi una str ...\n");
   scanf("%s", buffer);
                          /* ... e sua memorizzazione */
   v[i] = malloc(strlen(buffer)+1);
   if (v[i])
      strcpy(v[i], buffer);
   else {
      printf("eeekkk\n");
      break;
   }
 } /* fine for */

return;
}
```

abbiamo letto le stringhe da input e le abbiamo memorizzate, come stringhe esatte, nell'array di stringhe (cioe` puntatori) arrStr



Array di stringhe (ricerca) - 1/2 -

esercizio funzione "presenteIn" che

ricevendo una stringa **strCercata**, un array di stringhe, **char * v[N]**, la dimensione di v **dim**

restituisca **1 se strCercata e` in v, 0 altrimenti**

IL TIPO DI UN ARRAY DI STRINGHE

- un array di char e` ..... `char str[]`

equiv. (dal punto di vista dei tipi) a ..... `char *str`

- analogamente un array di stringhe di char e` `char *str[]`

equiv. (dal punto di vista dei tipi) a .....`char **str`

```
/* alg. di ricerca in array, con var. flag */
int presenteIn(
     char *strCercata, char **v, int dim) {

  int trovata, i;
              ...
```

se tutto quel che serve e` passare il parametro, va bene cosi`

memoria

Array di stringhe (ricerca) – 2/2 –

esercizio funzione che

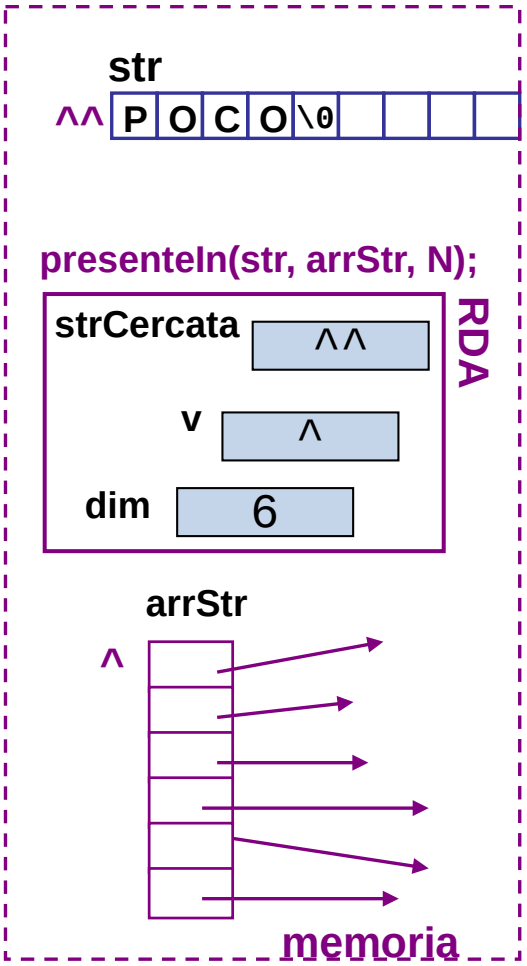
ricevendo una stringa **strCercata**, un array di stringhe, **char * v[N]**, la dimensione di v **dim**

restituisca **1 se strCercata e` in v, 0 altrimenti**

```
/* algoritmo di ricerca in array, con var. flag */

int presenteIn(
     char *strCercata, char **v, int dim) {

  int trovata, i;
```

3 osservazioni ☺

```
 for (i=0; (i<dim);  i++)
    if (strcmp(strCercata, v[i])==0)
       trovata=1;
return;
}
```



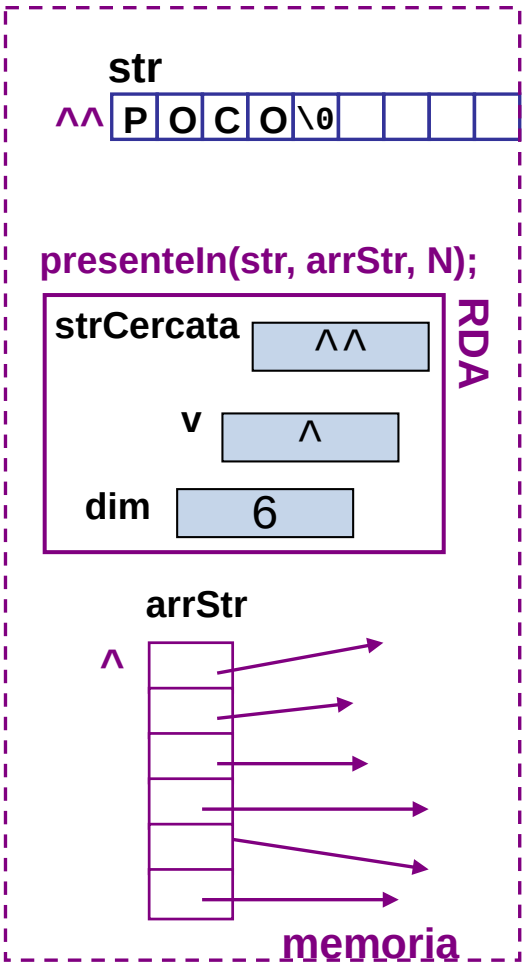
Array di stringhe (ricerca) – 2/2 –

esercizio funzione che

ricevendo una stringa **strCercata**, un array di stringhe, **char * v[N]**, la dimensione di v **dim**

restituisca **1 se strCercata e` in v, 0 altrimenti**

```
/* algoritmo di ricerca in array, con var. flag */

int presenteIn(
     char *strCercata, char **v, int dim) {

  int trovata, i;

  trovata = 0;        /* init flag (risultato che verra`
                      restituito se trovata non viene mai
                      modificata (strCercata mai trovata) */

 for (i=0; (i<dim) && !trovata;  i++)
    if (strcmp(strCercata, v[i])==0)
       trovata=1;
return trovata;
}
```

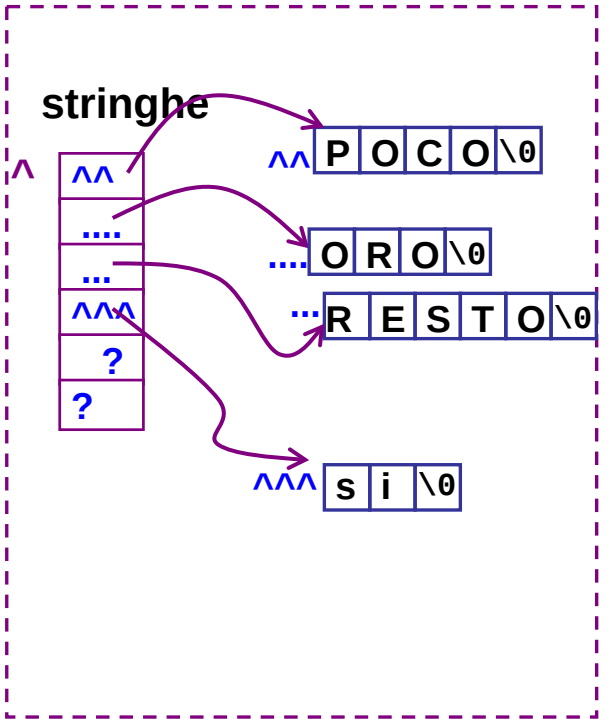


Programma gestione stringhe - introduzione

gestione di un array di al piu` N stringhe, ciascuna di al + LUNGMAX caratteri (array usato parzialmente)

Funzionalita` per la gestione di una *COLLEZIONE di stringhe:*

- **aggiunta** di una stringa (se possibile)
- **stampa** delle stringhe contenute
- **ricerca** di una stringa e rest. del suo indice (opp. -1)
  (*funzione di servizio*)
- **sostituzione** di una stringa con un'altra data



Quanto sopra e` parte della definizione di un *tipo di dati*
« collezione di stringhe »
(LE FUNZIONALITA`).

Riguardo alla STRUTTURA DATI?

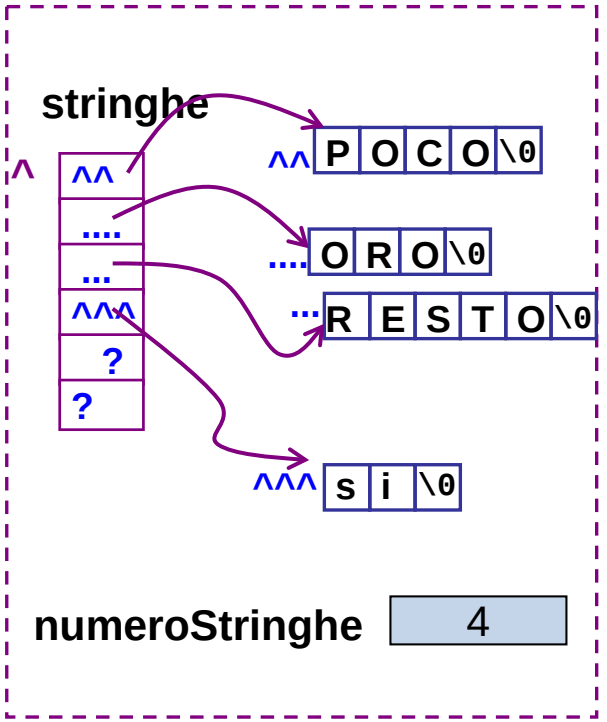
(come rappresentare questo oggetto in memoria?):

- **N ?**
- **sostegno per la collezione di stringhe? (Un array di stringhe)**
- **basta cosi`?**

Programma gestione stringhe - introduzione

gestione di un array di al piu` N stringhe, ciascuna di al + LUNGMAX caratteri (array usato parzialmente

Struttura dati e Funzionalita` per la gestione del TIPO
COLLEZIONE di stringhe



(come rappresentare questo oggetto in memoria?):

- **N e` una costante**
- **sostegno: l'array e` un array di N stringhe:**
  `char *stringhe[N]`

? Ma, se l'array e` usato parzialmente, dove fermare una scansione per stampa o ricerca? Dove inserire una nuova stringa?

stringhe e` quindi una variabile che va gestita usando anche l'**informazione addizionale** su
"quanti elementi/stringhe ci sono attualmente nell'array"

in sostanza una collezione di stringhe va rappresentata mediante la collaborazione di due variabili: **`stringhe`** e **`numeroStringhe`**

- un array di stringhe, che faccia da sostegno per la memorizzazione delle stringhe;
- una variabile intera che dica in ogni momento quante stringhe ci sono nell'array

collezione di stringhe = <array + numerostringhe>

Gestione di tabella (collezione) di stringhe - 1/8 -

SCHEMA DI PROGRAMMA

```
#include ...
#define N ...
... (dich.) ...
int main() {
  scelta      ----------(per il menu` ...)
  stringhe, numeroStringhe, ----------(per la collezione di stringhe)
  buffer1, buffer2,  ----------(buffer per leggere stringhe)
 do {
    /* ciclo di stampa menu`, lettura scelta funzionalita` da
    eseguire, esecuzione della funzionalita` prescelta */

    ... aggiungi(stringhe, buffer1, &numeroStringhe);   (scelta==1)

    ... stampaTutto(stringhe, numeroStringhe);          (scelta==3)

    ... sostituisci(stringhe, numeroStringhe, buffer1, buffer2);
                                                        (scelta==2)
    ...

 } while (scelta!=0)

return 0;
}
```

NB la coppia <stringhe, numeroStringhe> rappresenta la collezione di stringhe; collezione che a sua volta e` proprieta` della funzione main() ...

NB2 la struttura dati "tabella di stringhe" e` la coppia stringhe, numeroStringhe. Infatti sono quelle due componenti che permettono di gestirla. E infatti sono quelle due componenti che dobbiamo passare alle funzioni interessate.

Programma gestione stringhe - 2/8 - main()

```
#include <stdio.h>
#include <stdlib.h>          #define N ...          #define LUNGMAX
...    (dich.)
int main() { char *stringhe[N],     char buffer1[LUNGMAX+1],
             buffer2[LUNGMAX+1];
                                    int numeroStringhe, scelta;
 numeroStringhe = 0; /* init struttura dati array stringhe */

 do {  stampaMenu();                /* 1=aggiungi 2=sostitui... */
       scanf("%d", &scelta);              /* lettura scelta */
   switch(scelta) {
     case 1:  /* inserimento nuova stringa in stringhe oppure
                 messaggio di errore */
       break;
     case 2:   /* lett. stringa da sost. e sostituta; chiamata sostituisci() */
       break;
     case 3:   ...
```

```
switch(scelta) {
      case 1:
         codice da eseguire nel caso in cui scelta==1
      break;
  ...
      case VAL:
         codice da eseguire nel caso in cui scelta==VAL
      break;
  ...
      case VALVAL:
        codice da eseguire nel caso in cui scelta==VALVAL
      break;

default: printf(" scelta sbagliata \n\n");
} /* fine switch */
```

Programma gestione stringhe - 2/8 - main()

```
#include <stdio.h>
#include <stdlib.h>          #define N ...          #define LUNGMAX
...    (dich.)
int main() {  char *stringhe[N],     char buffer1[LUNGMAX+1],
              buffer2[LUNGMAX+1];
                                     int numeroStringhe, scelta;
 numeroStringhe = 0; /* init struttura dati array stringhe */

 do {  stampaMenu();                 /* 1=aggiungi 2=sostitui... */
       scanf("%d", &scelta);              /* lettura scelta */
    switch(scelta) {
      case 1:  ... aggiungi(stringhe, buffer1, &numeroStringhe);
        break;
      case 2: ... sostituisci(stringhe, numeroStringhe, buffer1,
                                                        buffer2);
        break;
      case 3:  stampaTutto(stringhe, numeroStringhe); break;
      case 0:  printf("FINE PROGRAMMA\n"); break;
      default: printf(" scelta sbagliata \n\n");
    } /* fine switch */
} while (scelta!=0)
return 0;
}
```

NB stampaTutto riceve “la collezione”, sotto forma di una coppia di parametri

Programma gestione stringhe - 3/8 - main()

```
#include <stdio.h>
#include <stdlib.h>                #define N ...                        #define LUNGMAX
...     (dich.)
int main() {            char *stringhe[N],    char buffer1[LUNGMAX+1],            buffer2[LUNGMAX+1];
                        int numeroStringhe, scelta;

 numeroStringhe = 0; /* init struttura dati array stringhe */

 do {  stampaMenu();                    /* 1=aggiungi 2=sostitui... */
       scanf("", &scelta);              /* lettura scelta */

   switch(scelta) {
     case 1:
       if (numeroStringhe<N) {
         printf("quale stringa da aggiungere? ");
          scanf("%s", buffer1);
         aggiungi(stringhe, buffer1, &numeroStringhe);
       }
       else  printf("spazio insufficiente, tsk.\n\n");
       break;

    case 2:
          printf("stringa da sostituire: ");
     scanf("%s", buffer1);
          printf("stringa con cui sostituire: ");
     scanf("%s", buffer2);
          sostituisci(stringhe, numeroStringhe, buffer1, buffer2);
          break;
```

Controllo se c'e` spazio per una nuova stringa, nell'array sostegno

NB aggiungi riceve "la collezione", sotto forma di una coppia di parametri

Programma gestione stringhe - 3/8 - main()

```
#include <stdio.h>
#include <stdlib.h>                #define N ...                        #define LUNGMAX
...     (dich.)
int main() {            char *stringhe[N],    char buffer1[LUNGMAX+1],            buffer2[LUNGMAX+1];
                        int numeroStringhe, scelta;

 numeroStringhe = 0; /* init struttura dati array stringhe */

 do {  stampaMenu();                    /* 1=aggiungi 2=sostitui... */
       scanf("", &scelta);              /* lettura scelta */

   switch(scelta) {
     case 1:
       if (numeroStringhe<N) {
         printf("quale stringa da aggiungere? ");
          scanf("%s", buffer1);
         aggiungi(stringhe, buffer1, &numeroStringhe);
       }
       else  printf("spazio insufficiente, tsk.\n\n");
       break;

    case 2:
          printf("stringa da sostituire: ");
      scanf("%s", buffer1);
          printf("stringa con cui sostituire: ");
      scanf("%s", buffer2);
          sostituisci(stringhe, numeroStringhe, buffer1, buffer2);
          break;
```

perche'?

NB aggiungi riceve “la collezione”, sotto forma di una coppia di parametri

Programma gestione stringhe - 4/8 - main()

```
#include <stdio.h>
#include <stdlib.h>                #define N ...                        #define LUNGMAX
...     (dich.)
int main() {            char *stringhe[N],    char buffer1[LUNGMAX+1],            buffer2[LUNGMAX+1];
                        int numeroStringhe, scelta;

 numeroStringhe = 0; /* init struttura dati array stringhe */

 do {  stampaMenu();                    /* 1=aggiungi 2=sostitui... */
       scanf("", &scelta);              /* lettura scelta */

   switch(scelta) {
     case 1:
        if (numeroStringhe<N) {
          printf("quale stringa da aggiungere? ");
     scanf("%s", buffer1);
          aggiungi(stringhe, buffer1, &numeroStringhe);
        }
        else  printf("spazio insufficiente, tsk.\n\n");
        break;

     case 2:
       printf("stringa da sostituire: ");
       scanf("%s", buffer1);
       printf("stringa con cui sostituire: ");
       scanf("%s", buffer2);
       sostituisci(stringhe, numeroStringhe, buffer1, buffer2);
      break;
...
```

NB sostituisci riceve “la collezione”, sotto forma di una coppia di parametri

Programma gestione stringhe - 5/8 -

stampaTutto()

```
...
case 3:  stampaTutto(stringhe, numeroStringhe);
         break; ...
```

```
void stampaTutto(char *v[], int quanteSono) {
  int i;

for (i=0; i<quanteSono; i++)
   printf("%s\n", v[i]);
return;                    /* o anche *(v+i) */
}
```

- a che tipo e' equivalente char *v[] (solo dal punto di vista dei tipi nei parametri)

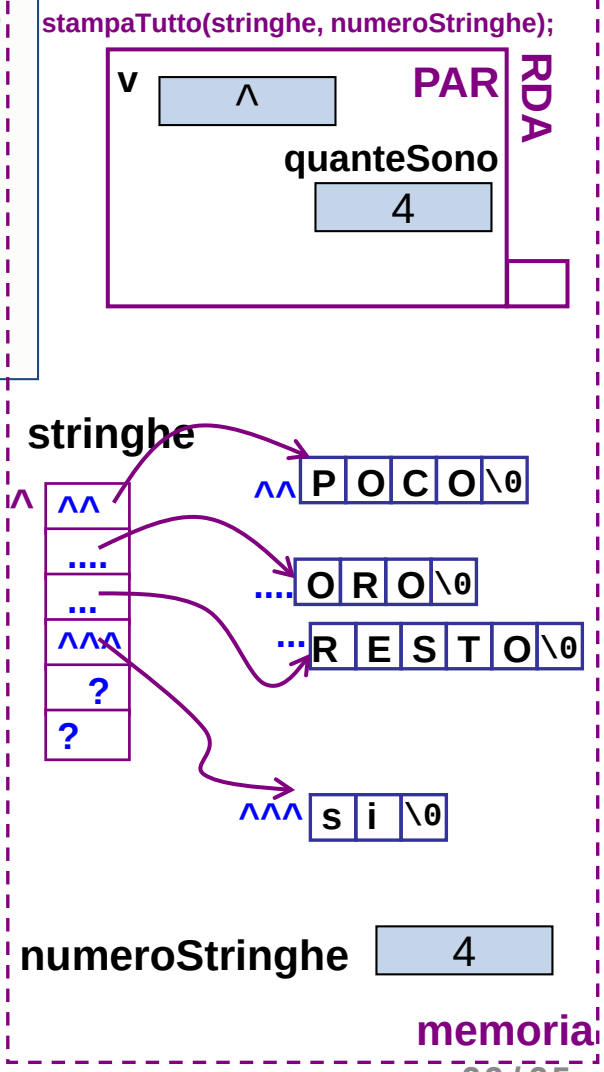
- v[i] e` il ..... alla ...-esima ..... di v

- v[i] si puo` scrivere anche come .....

- cosa e` v, tra le scelte seguenti?

  "doppio puntatore", "puntatore a puntatore", indirizzo di un puntatore, indirizzo di una locazione che contiene un ind.

- cosa vuol dire "stampare v[i] con formato %s"

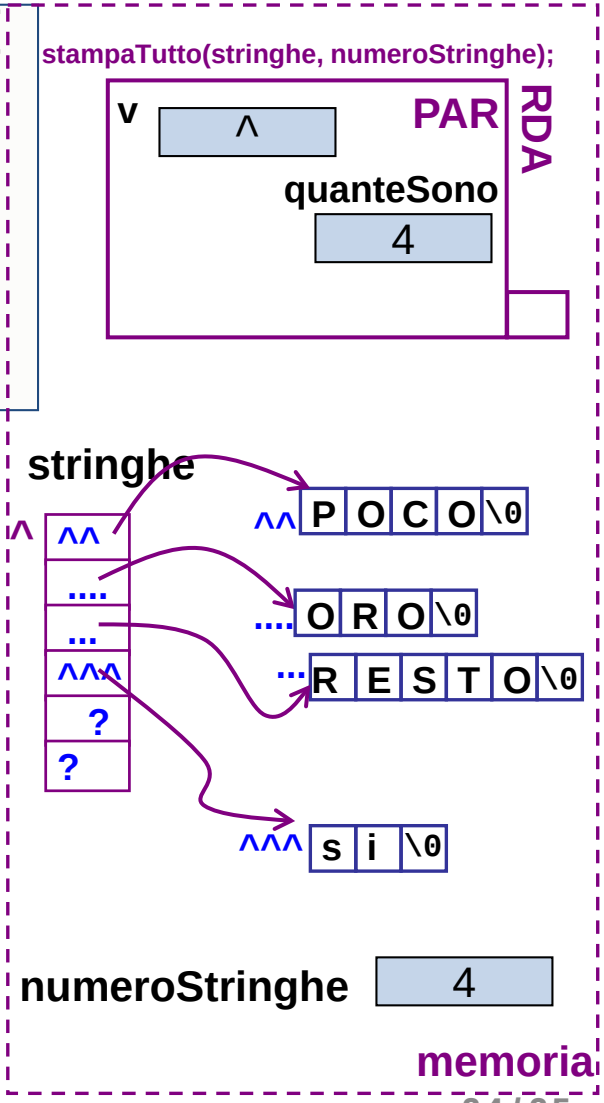


Programma gestione stringhe - 5/8 -

stampaTutto()

```
...
case 3:  stampaTutto(stringhe, numeroStringhe);
         break; ...
```

```
void stampaTutto(char *v[], int quanteSono) {
  int i;

for (i=0; i<quanteSono; i++)
   printf("%s\n", v[i]);
return;                    /* o anche *(v+i) */
}
```



- char *v[] equivalente a char **
- v[i] = puntatore alla i-esima stringa = *(v+i);
- v e` "doppio puntatore"
  = "puntatore a puntatore"
  = indirizzo di un puntatore
  = ind. di una locazione che contiene un ind.
- stampare v[i] con formato %s vuol dire stampare la stringa v[i], cioe` la stringa puntata dal puntatore v[i]

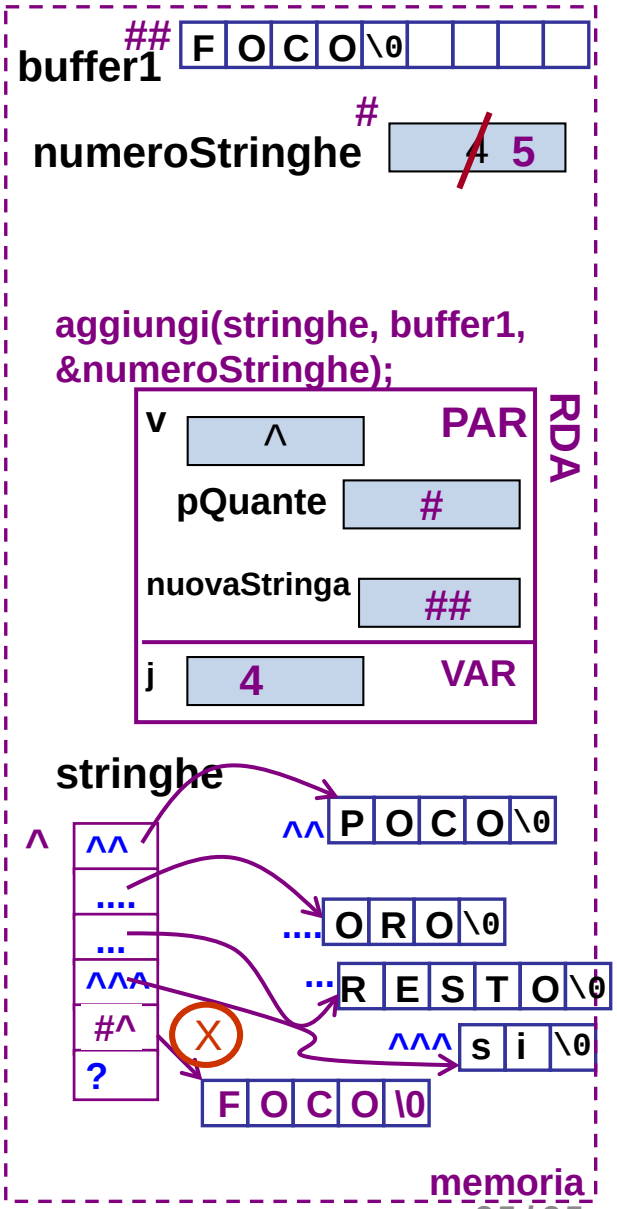
Programma gestione stringhe - 6/8 - aggiungi()

```
void aggiungi ( char **v, char *nuovaStringa, int *pQuante) {
/* il controllo sulla disponibilita` di spazio nell'array
si suppone fatto all'esterno, dalla funzione chiamante
(non e` bello, ma ora ci stiamo concentrando su altro */

  int j = *pQuante; /* solo per comodita` */

 v[j] = malloc (strlen(nuovaStringa)+1);

 if (!v[j])
    printf("errore in alloc. ...");
 else {
   strcpy(v[j], nuovaStringa);
   *pQuante+=1;
 }
return;
}
```

X

ALCUNE VERITa`

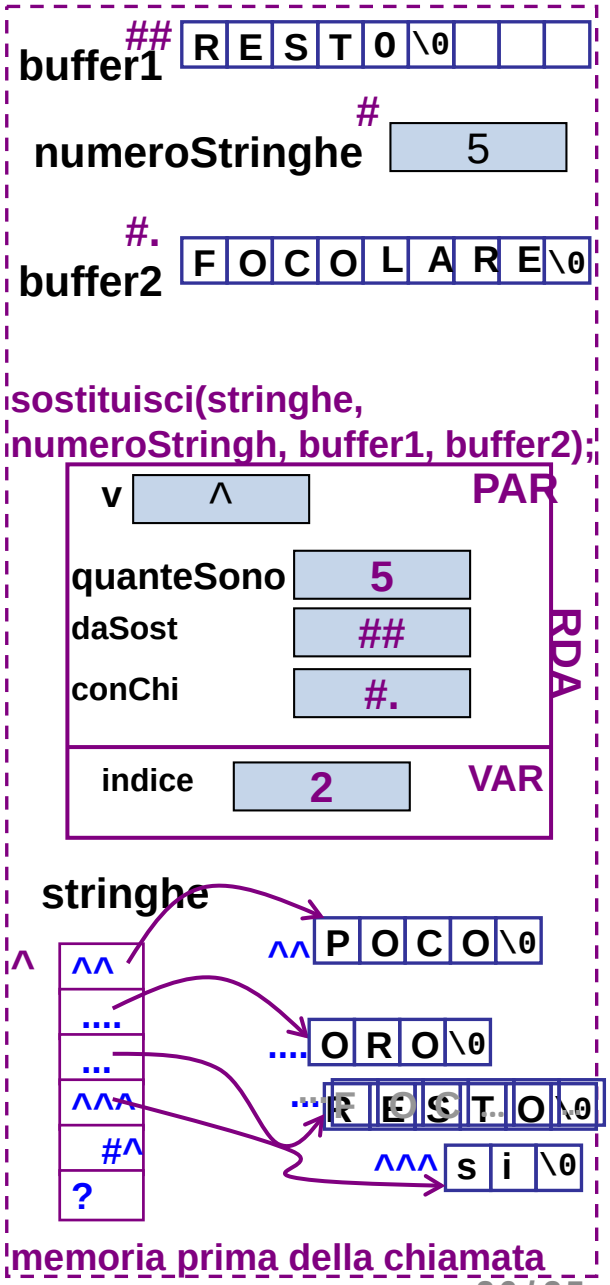
- la funzione ha aggiunto una stringa in posizione numeroStringhe+1; quindi subito prima del termine dell'attivazione, numeroStringhe viene incrementato di 1;
- l'espressione (!v[j]) e` equiv. a (v[j]==NULL)



Programma gestione stringhe - 7/8 -

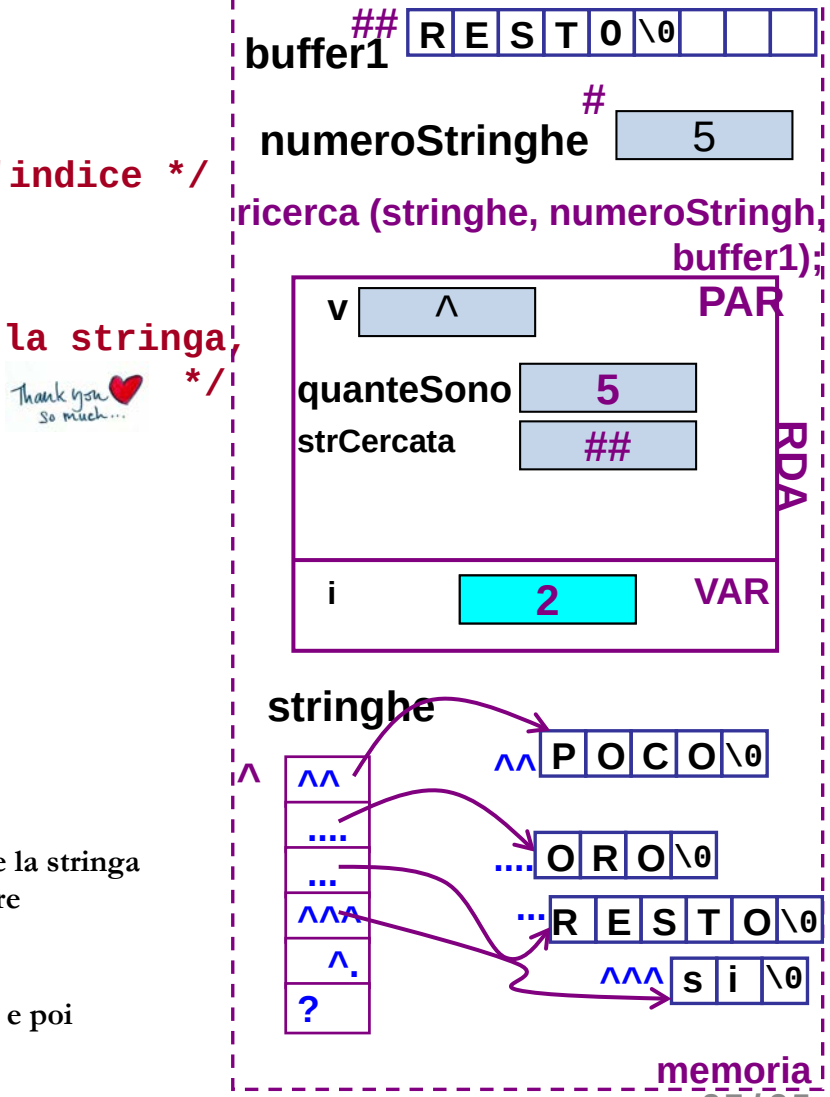
sostituisci()

```
void sostituisci (char **v, int quanteSono,
                  char *daSost, char *conChi) {
/* cerchiamo l'indice della stringa da sostituire con
   una funzione di servizio che restituisce l'indice
   della stringa nell'array, oppure -1 (se non c'e`)*/
  int indice =
         ricerca(v,quanteSono, daSost);

 if (indice==-1)
    printf("non presente\n\n");
 else {
    free(v[indice]; /* deall. stringa da sost.
                    e allocazione stringa sostituto */
    v[indice]=malloc(strlen(conChi)+1);

    if (!v[indice])
       printf("errore in alloc. ...");
    else
       strcpy(v[indice], conChi);

 } /* fine primo if */
return;
}
```



Programma gestione stringhe - 8/8 - ricerca()

```
int ricerca (char **v, int quanteSono, char *strCercata) {
/* restituisce l'indice di strCercata in v, oppure  -1 */
  int i = 0,

 for ( ; i<quanteSono; i++)
   if (strcmp((v[i], strCercata)==0)
     return i; /* stringa trovata: rest. l'indice */


/* se siamo usciti dal ciclo senza mai trovare la stringa,
 ... vuol dire che non l'abbiamo trovata ...      */
return -1;
}
```



esercizio: riflettere sul perche', nella funzione sostituisci, per sostituire la stringa v{indice] con quella conChi, dopo `free(v[indice])`, invece di fare

`v[indice] = conChi`

abbiamo usato codice differente per creare una copia esatta di conChi e poi assegnare a v[indice] tale nuova stringa

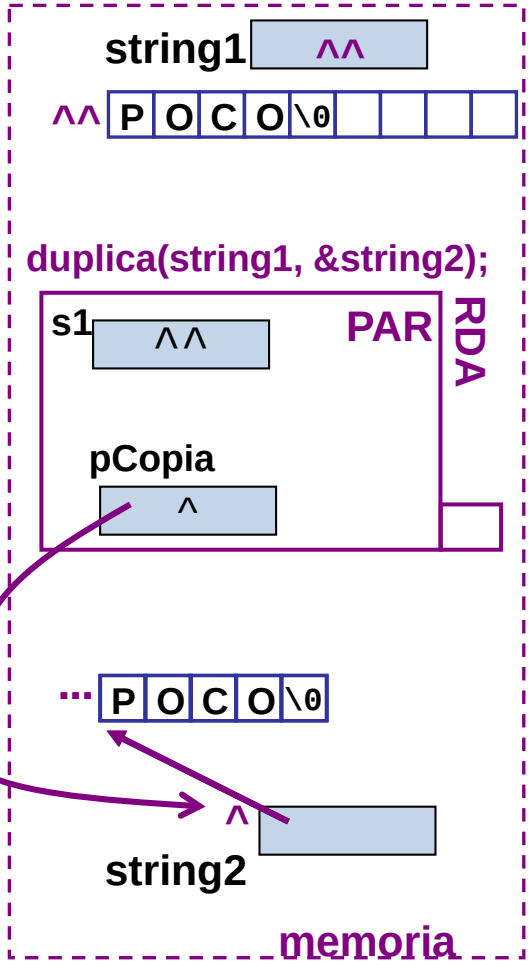
Esercizio (duplica stringa)

programma che esegue una duplicazione di stringa mediante side effect da parte della funzione duplica()

```
#include <stdio.h>
#include <stdlib.h>
     void duplica(char *, char **);
int main() { char *string1, *string2;
   ...
/* string1 e` una stringa effettiva; string2 e` un
puntatore cui attacchiamo un duplicato della string1 */
...
   duplica(string1, &string2);
...
return 0;
}

void duplica(char * s1, char **pCopia) {
   *pCopia = malloc(strlen(s1)+1);
   if (*pCopia)
      strcpy(*pCopia, s1);
return;
}
```

Questo parametro attuale e` un "indirizzo di locazione capace di contenere un indirizzo" *(l'indirizzo di un indirizzo …)*

Questo parametro formale e` capace di ricevere un valore che e` indirizzo di un indirizzo di carattere



osservazione: Cosa c'e` in *pCopia se l'allocazione e` andata male?

Ora rispondi e poi fai una funzione che duplica come sopra ma restituisce 1/0 per indicare il successo dell'operazione. Poi prosegui

duplica2

funzione come duplica(), che restituisce 1 o 0 a seconda del successo dell'operazione di duplicazione

```
int duplica2(char * s1, char **pCopia) {
  *pCopia = malloc(strlen(s1)+1);
  if (*pCopia) {
    strcpy(*pCopia, s1);
    return 1;       /* e` andata bene */
  } else
    return 0;       /* e` andata male */
}
```

Esercizio

programma che legge un intero n e poi legge n double;
memorizza i double in un array dinamico esatto,
calcola e stampa minimo, massimo e media dei double

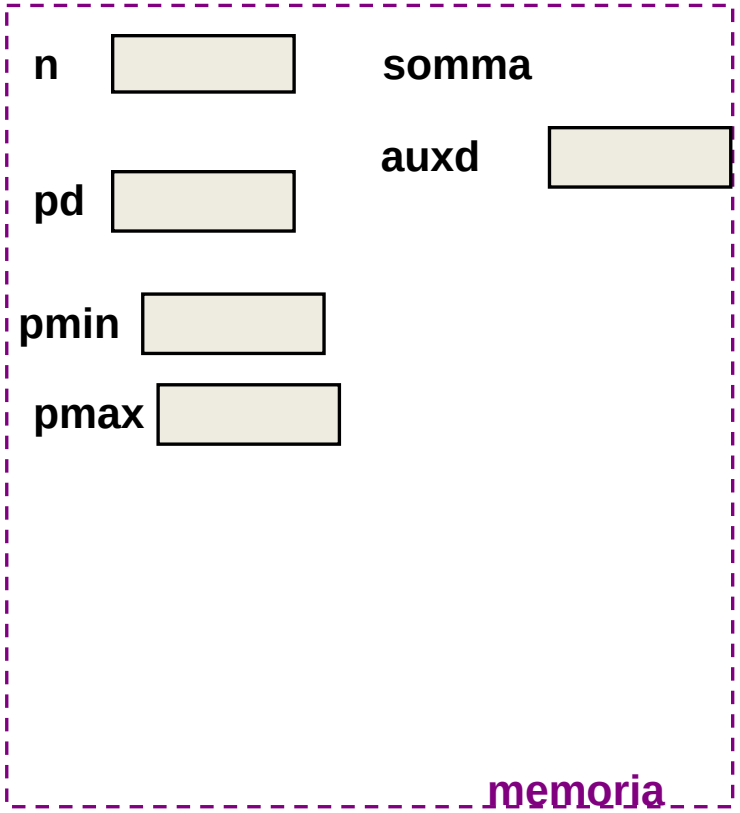
1) Allocazione array dinamico, lettura e memorizzazione dei numeri negli elementi *pd ..... *(pd+n-1)

2) init minimo e massimo parziale, e somma

3) scansione a ritroso da "**penultimo**" a "primo" elemento, usando l'algoritmo del massimo (minimo) parziale
e accumulando i double (per poter calcolare la media)

4) e poi calcolo media e stampa di min, max e media

La scansione viene realizzata mediante un puntatore: auxd



Esercizio

programma che legge un intero n e n double;
li memorizza in un array dinamico esatto
calcola e stampa minimo, massimo e media dei double

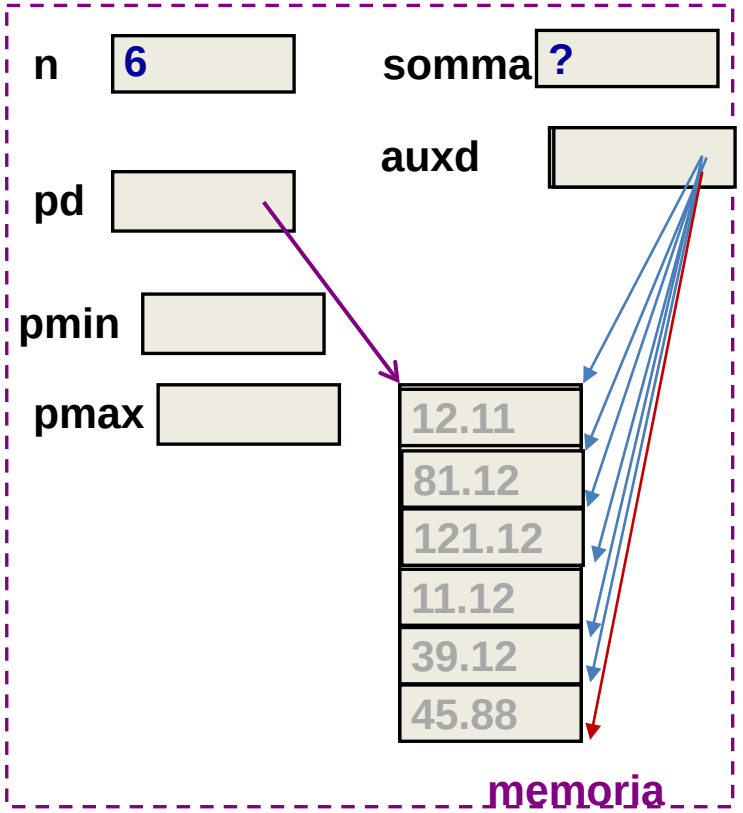
1) Allocazione array dinamico, lettura e memorizzazione dei numeri negli elementi *pd ..... *(pd+n-1)

2) init minimo e massimo parziale, e somma

3) scansione a ritroso da "**penultimo**" a "primo" elemento, usando l'algoritmo del massimo (minimo) parziale
e accumulando i double (per poter calcolare la media)

4) e poi calcolo media e stampa di min, max e media

La scansione viene realizzata mediante un puntatore: auxd



Esercizio (o esempio?)

programma che legge un intero n e n double;
li memorizza in un array dinamico esatto
calcola e stampa minimo, massimo e media dei double

1) Allocazione array dinamico, lettura e memorizzazione dei numeri in *pd ..... *(pd+n-1)

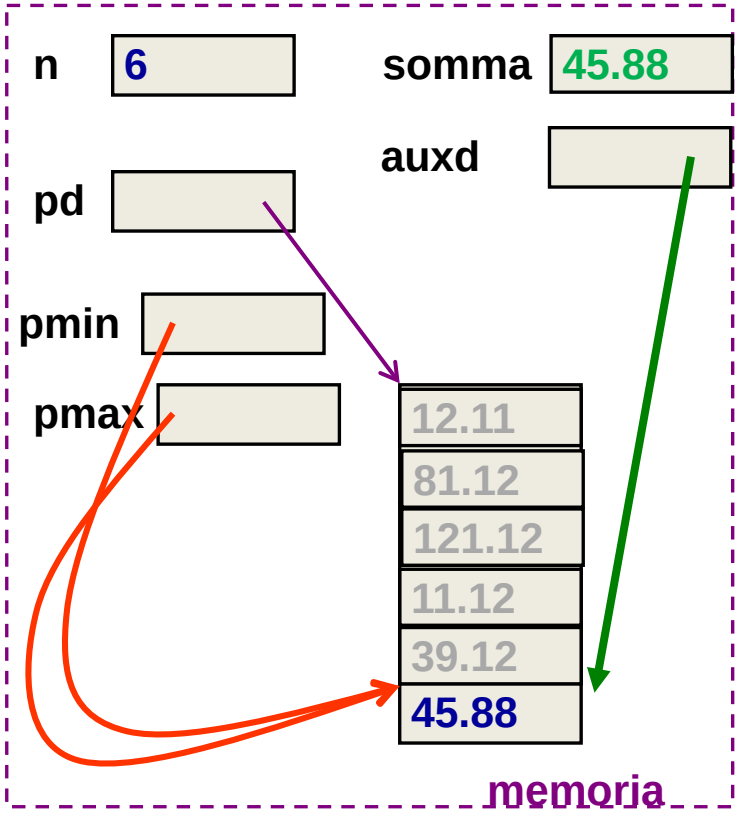
2) init minimo e massimo parziale, e somma

3) scansione a ritroso da "**penultimo**" a "primo" elemento, trovando max e min, e accumulando

4) e poi calcolo media

MA usiamo (per realizzare l'alg. di massimo/minimo parziale)

- indirizzo del max parz: pmax
- indirizzo del min parz: pmin
- scansione degli elementi con un puntatore: auxd



Esercizio

programma che legge un intero n e n double;
li memorizza in un array dinamico esatto
calcola e stampa minimo, massimo e media dei double

1) Allocazione array dinamico, lettura e memorizzazione dei numeri in *pd ..... *(pd+n-1)

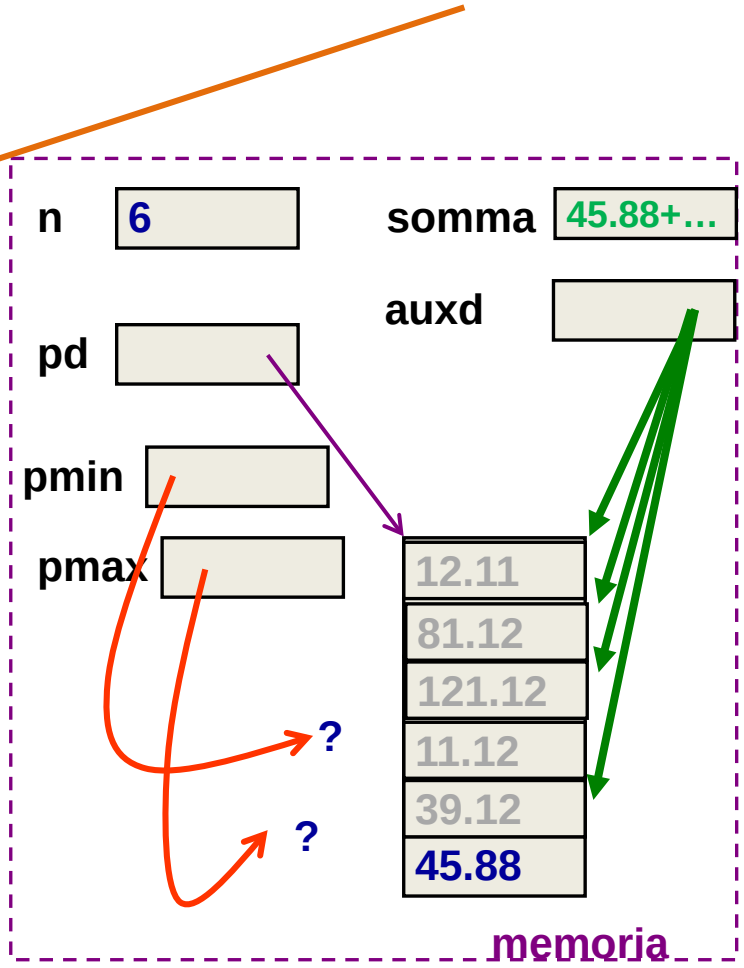
2) init minimo e massimo parziale, e somma

3) scansione a ritroso da "**penultimo**" a "primo" elemento, trovando max e min, e accumulando

4) e poi calcolo media

MA usiamo (per realizzare l'alg. di massimo/minimo parziale)

- indirizzo del max parz: pmax
- indirizzo del min parz: pmin
- scansione degli elementi con un puntatore: auxd
  - se *auxd e` maggiore di *pmax, allora *auxd e` un nuovo max parz: pmax = auxd



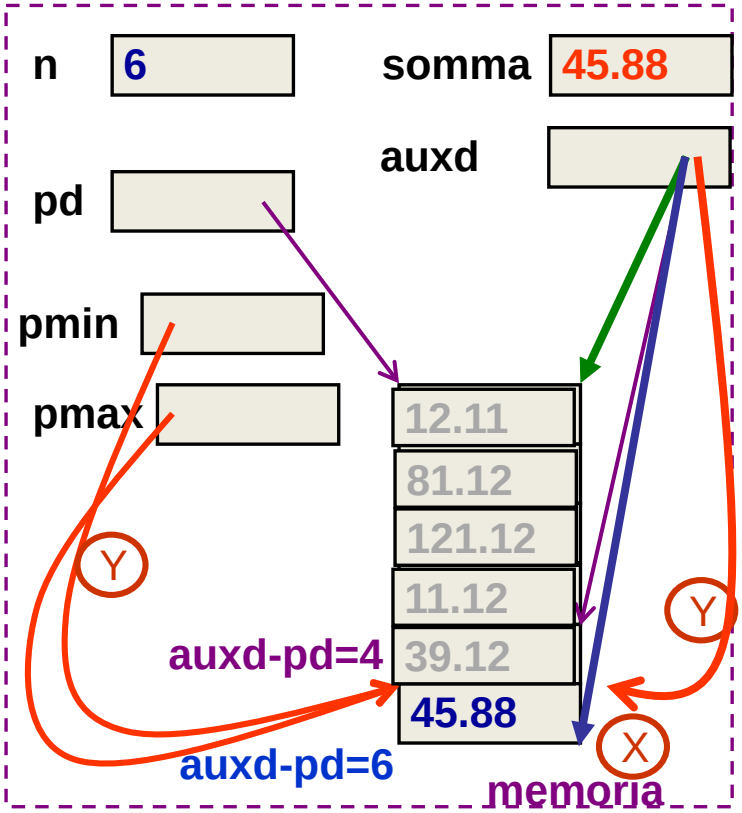
esercizio su intero n e n double (coding 1/2)

programma che legge un intero n e n double;
li memorizza in un array dinamico esatto
calcola e stampa minimo, massimo e media dei double

```
#include <stdio.h>
#include <stdlib.h>
int main() {
...
```

```
int n;
double *pd, *pmax, *pmin, *auxd, somma,   ;
```

```
  scanf( ... &n);
  pd = malloc(n*sizeof(double));

  if (!pd)   printf(" ... ");
  else {
    for (auxd=pd; auxd-pd < n; auxd++)
      scanf("%lf", auxd);

    /* inizializzazione: pmax e pmin saranno
       i puntatori al massimo e minimo;
       tecnica del massimo parziale */
    pmax = pmin = --auxd;
    somma = *auxd;
    ...
```



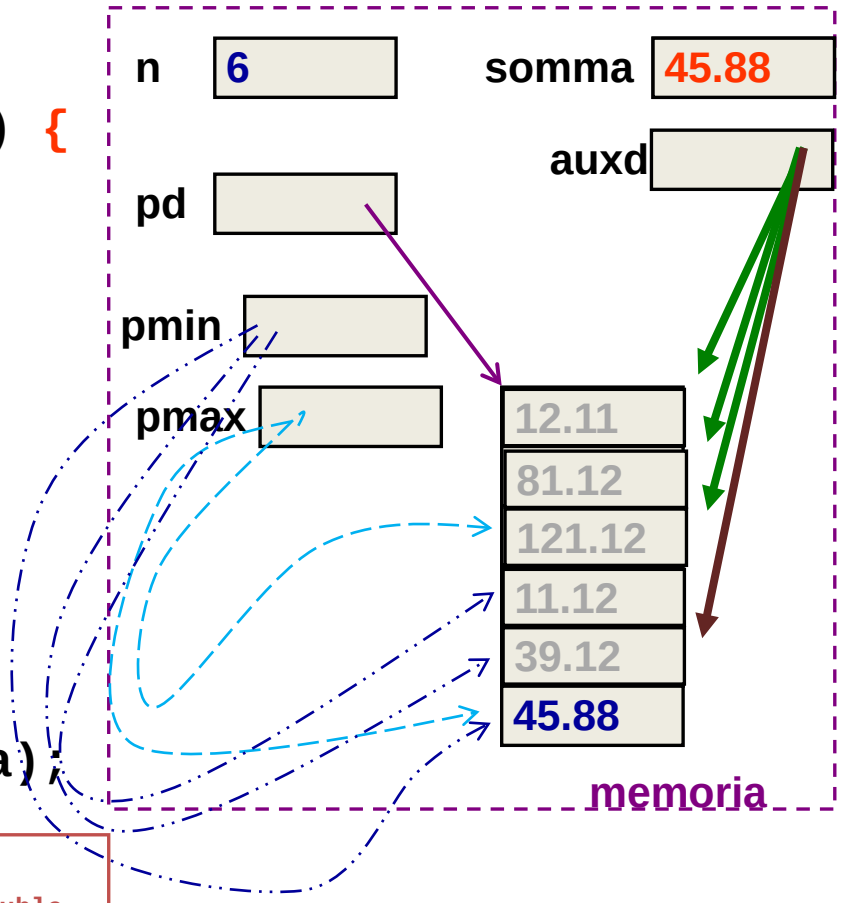
esercizio su intero n e n double (coding 2/2)

```
...  pmax = pmin = --auxd;
     somma = *auxd;
     for (auxd--; auxd >= pd; auxd--) {
       if (*pmax < *auxd)
         pmax=auxd;

       if (*pmin > *auxd)
         pmin=auxd;

       somma += *auxd;
     }

     media = somma/n;
     printf( ..., *pmax, *pmin, media);
return 0;
}
```

espressione double (divisione tra un double e un intero

undefined ... nella pagina prima;

Deve esprimere la media, cioe` un valore double



auxd viene inizialmente retrocesso all'inizio della componente n-esima (indice n-1); poi, mentre si mantiene >=pd si decrementa per toccare tutte le altre componenti dell'array, in ordine inverso (indice n-2, n-3, ... 0).

Per ogni componente toccata (indicata) da auxd, si attua la tecnica di mantenimento del massimo (e minimo) parziale (*auxd e` il contenuto della locazione puntata da auxd), e la si somma nell'accumulatore (somma=somma+ *auxd)