Tecniche della Programmazione, lez.14

- puntatori
- array e puntatori … discussione di alcuni esercizi dati

- Richiamo su RDA (perche' servono molto da qui in poi)
- Varie applicazioni dell'uso di puntatori
  - Scansione e ricerca in array
  - Riuso di una funzione su array …
- Uso di puntatori per gestire parametri di output

Chiamata ed Attivazione di funzione – 1/2 –

```
int mcd(int n, int m) {
  int risult;
  while (n!=m)
     if (m>n)
       m=m-n;
     else n-=m;
  risult = n;   /* (o m ...) */
return risult;
}

#include <stdio.h>
int main() {
  int primo, secondo,
      dummy=13, pfui=2;

 printf("...numeri... ");
 scanf("%d %d", &primo, &secondo);

 printf("mcd=%d\n", mcd(primo/2*pfui, secondo-dummy+2+11));
return 0;
}
```

funzione mcd,

PARAMETRI FORMALI: n, m

VARIABILI LOCALI: risult

VALORE RISULTATO di tipo int

CHIAMATA di funzione,

- PARAMETRI ATTUALI: ESPRESSIONI

ATTIVAZIONE

1) passaggio dei parametri: ATTUALI ---> FORMALI

2) esecuzione del codice della funzione

3) restituzione del risultato

Chiamata ed Attivazione di funzione – 2/2 –

```
#include <stdio.h>
int main() {
  int primo, secondo ...;
 ...
 printf("mcd=%d\n", mcd(primo/2*pfui, secondo-dummy+2+11));
return 0;
}
```

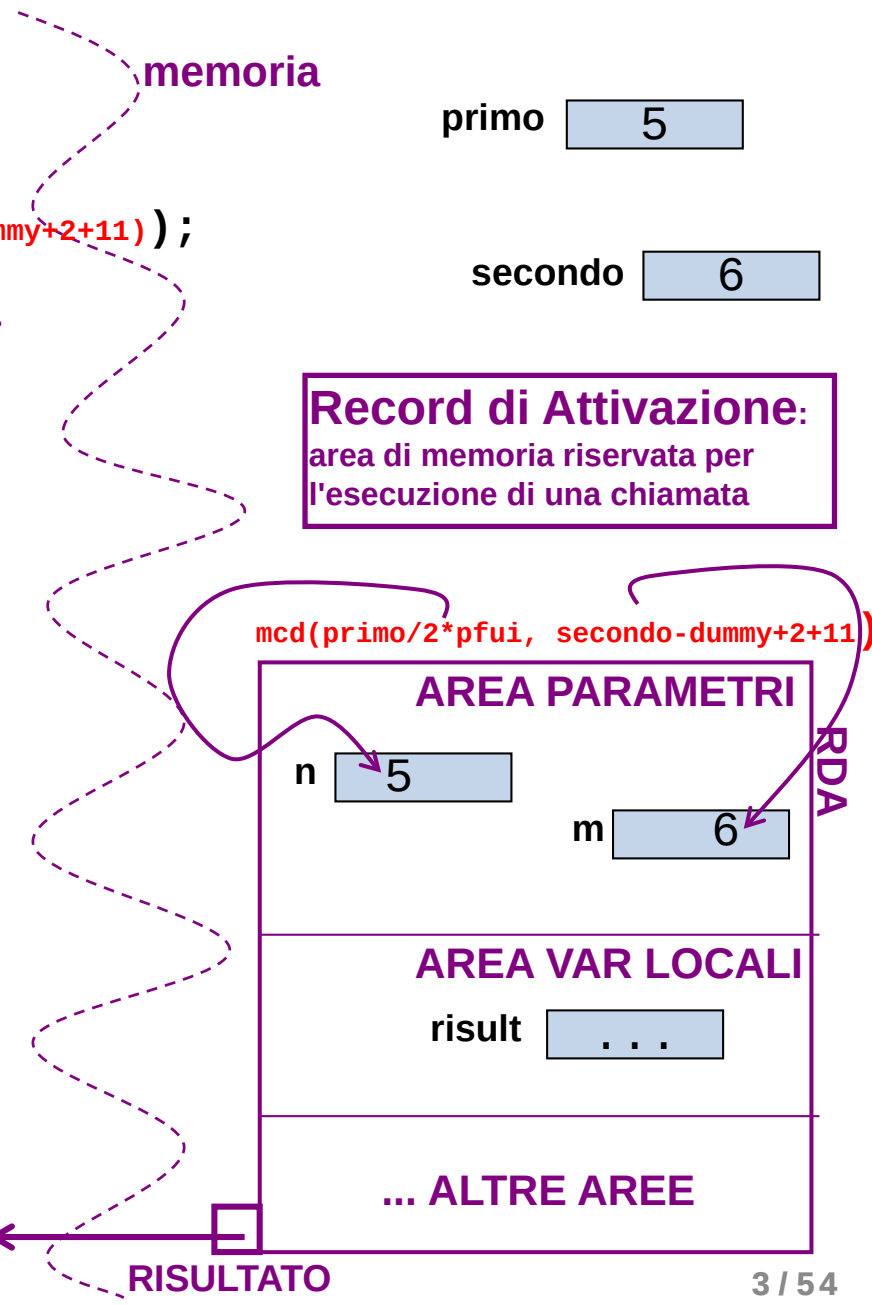
CHIAMATA di funzione,

- **PARAMETRI ATTUALI:** **ESPRESSIONI**

ATTIVAZIONE

1) passaggio dei parametri: **ATTUALI ---> FORMALI**
2) esecuzione del codice della funzione
3) restituzione del risultato

```
int mcd(int n, int m) {
  int risult;
...
return risult;
}
```



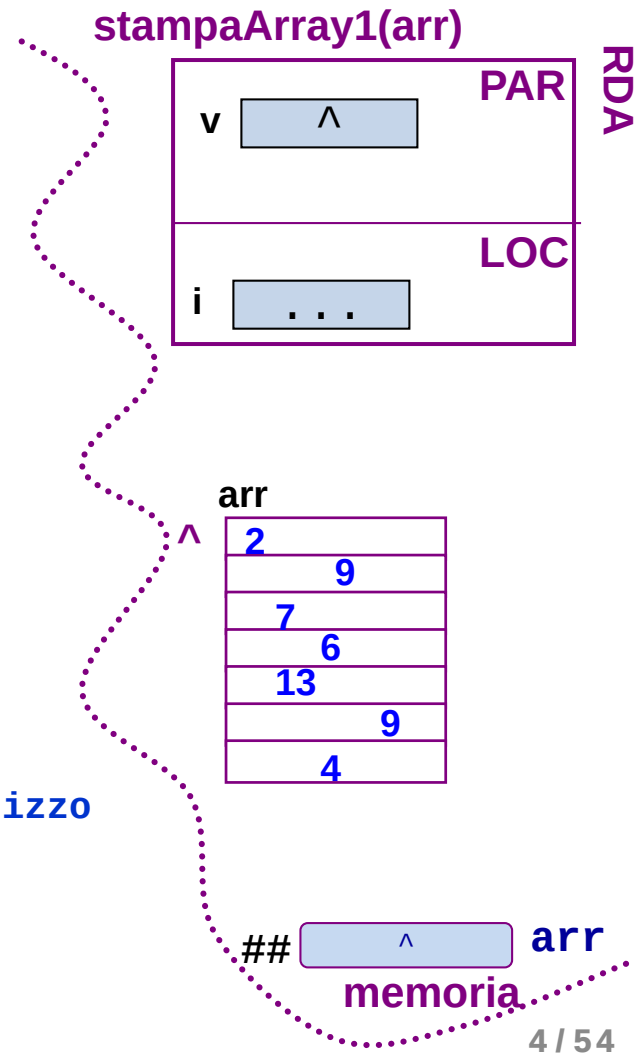
Scansione array in vari modi - 1/4 -

esercizio funzione che
ricevendo un array (**v**) di N interi, (noto N come costante)
stampa v

```
#define N 7
...
void stampaArray1 (int v[N]) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, v[i]);
return;
}


int main() {
  int arr[N] = {...};
...
stampaArray1(arr);
...
...
}
```

stiamo passando un indirizzo

chiamata



Scansione array in vari modi – 2/4 –

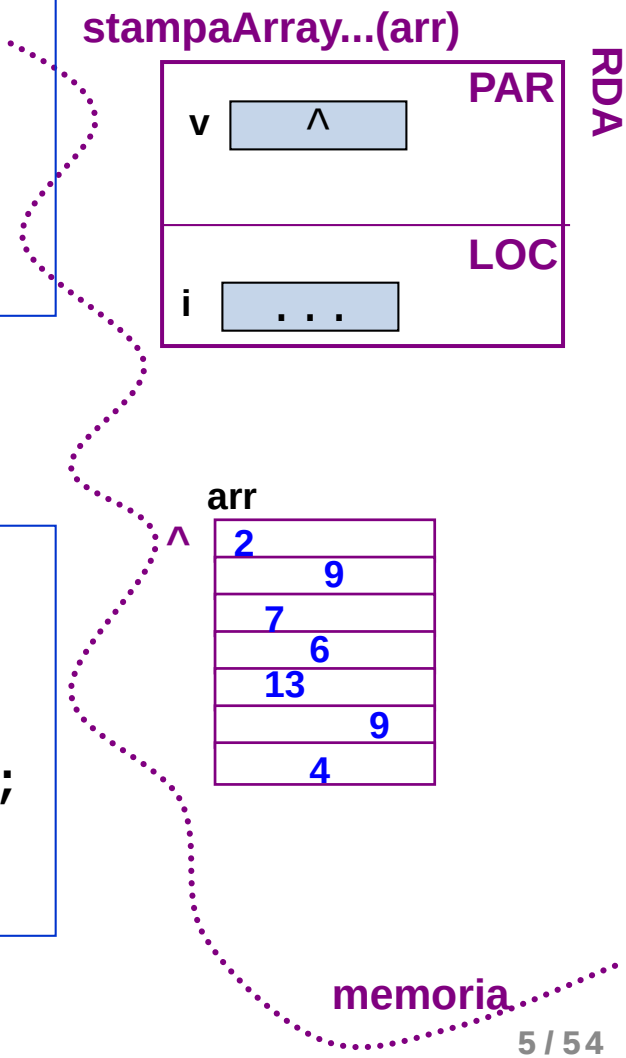
come sopra, ma il codice che usa l'array lo usa nella sua veste di puntatore

```
void stampaArray1 (int v[N]) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, v[i]);
return;
}
```

CHIAMATA: stampaArray1(arr)

```
void stampaArray2 (int v[N]) {
  int i;

                                   v[i] ≡ *(v+i)
 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, *(v+i));
return;
}
```

CHIAMATA: stampaArray2(arr)



Scansione array in vari modi - 3/4 -

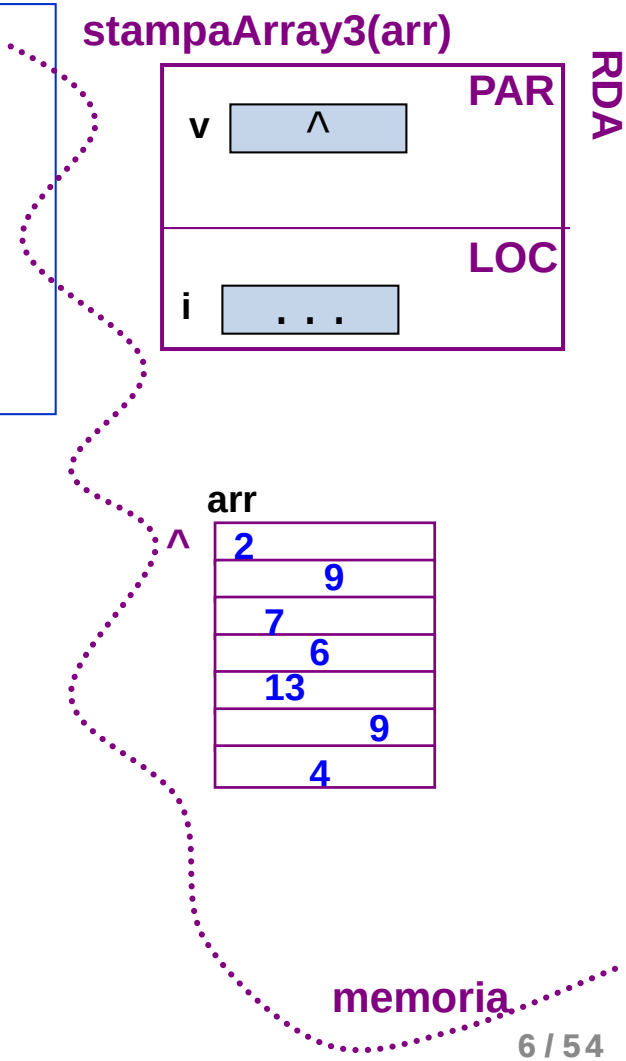
come sopra, ma stampaArray3 e 4 hanno un parametro puntatore: dato che quel che si passa nella chiamata e` un puntatore, non c'e` problema.
NB in stampaArray4 usiamo un puntatore come se fosse il nome di un array ...

```
void stampaArray3 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf … … … …
return;
}
```

`v[i] ≡ *(v+i)`

☺ cosa?

`CHIAMATA: stampaArray3(arr)`

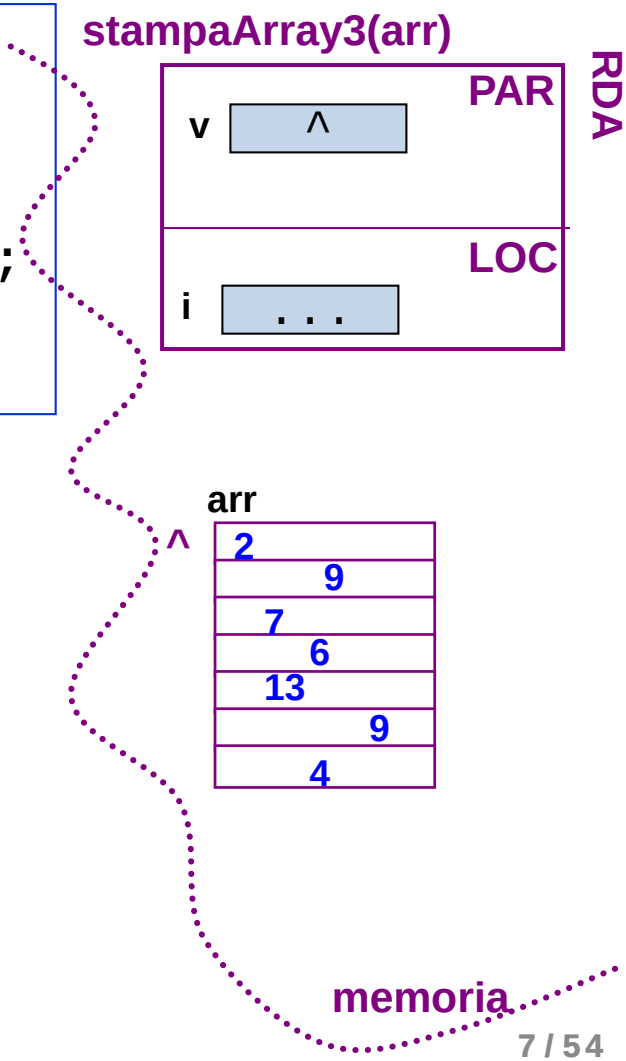


Scansione array in vari modi - 3/4 -

come sopra, ma stampaArray3 e 4 hanno un parametro puntatore: dato che quel che si passa nella chiamata e` un puntatore, non c'e` problema.
NB in stampaArray3 usiamo un puntatore come se fosse il nome di un array ...

```
void stampaArray3 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, *(v+i));
return;
}
```

CHIAMATA: stampaArray3(arr)



Scansione array in vari modi - 3/4 -

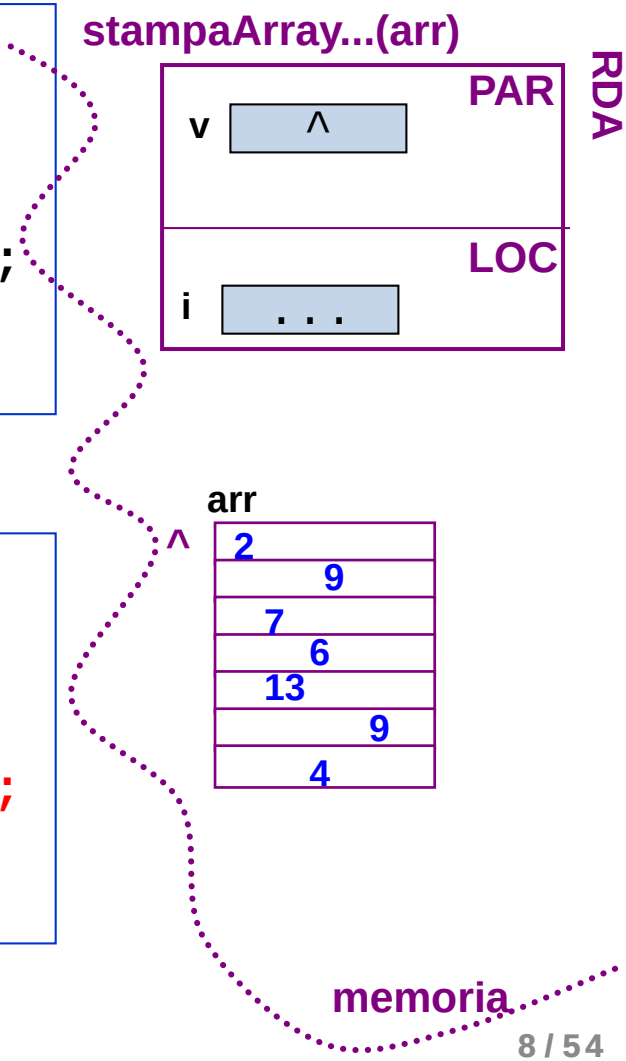
come sopra, ma stampaArray3 e 4 hanno un parametro puntatore: dato che quel che si passa nella chiamata e` un puntatore, non c'e` problema.
NB in stampaArray3 usiamo un puntatore come se fosse il nome di un array ...

```
void stampaArray3 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, *(v+i));
return;
}
```

CHIAMATA: stampaArray3(arr)

```
void stampaArray4 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, ??????);
return;
}
```

CHIAMATA: stampaArray4(arr)



Scansione array in vari modi - 3/4 -

come sopra, ma stampaArray3 e 4 hanno un parametro puntatore: dato che quel che si passa nella chiamata e` un puntatore, non c'e` problema.
NB in stampaArray3 usiamo un puntatore come se fosse il nome di un array ...

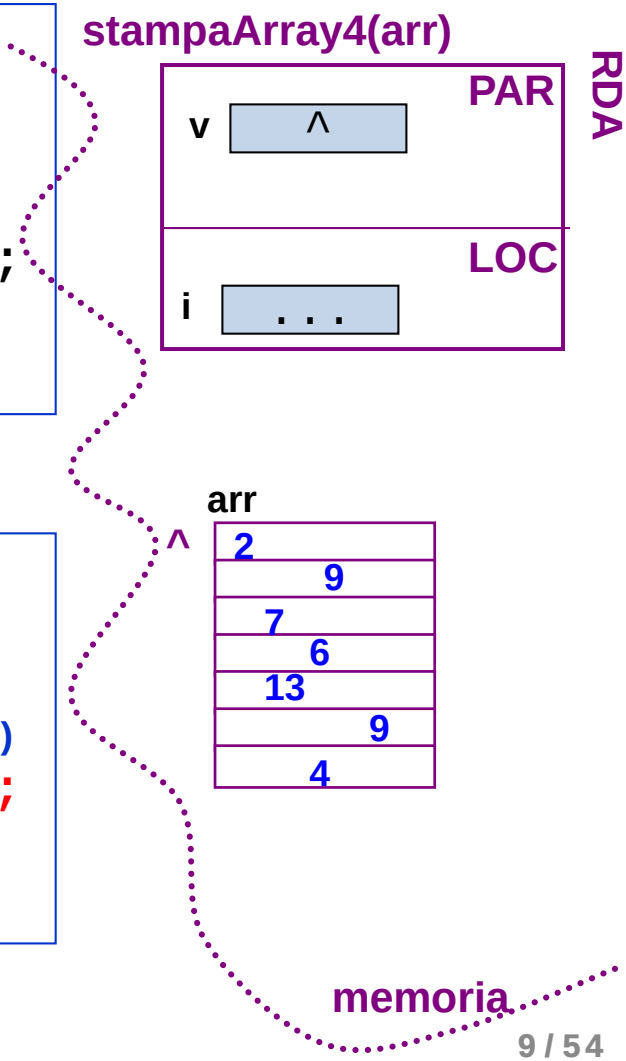
```
void stampaArray3 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, *(v+i));
return;
}
```

CHIAMATA: stampaArray3(arr)

```
void stampaArray4 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, ??????);
return;
}
```

v[i] ≡ *(v+i)

CHIAMATA: stampaArray4(arr)



Scansione array in vari modi - 3/4 -

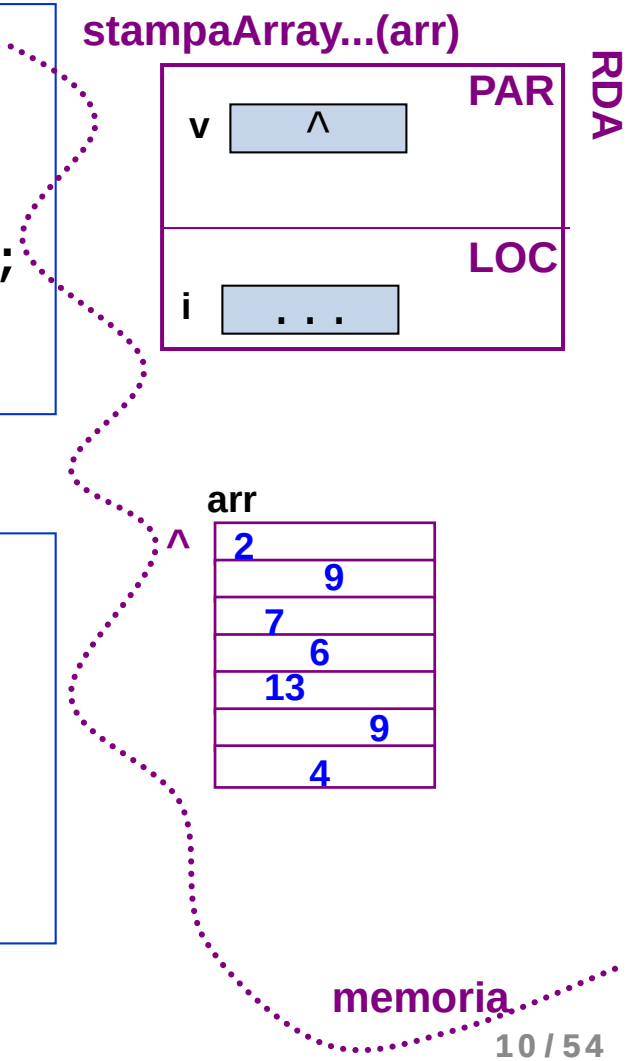
come sopra, ma stampaArray3 e 4 hanno un parametro puntatore: dato che quel che si passa nella chiamata e` un puntatore, non c'e` problema.
NB in stampaArray3 usiamo un puntatore come se fosse il nome di un array ...

```
void stampaArray3 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, *(v+i));
return;
}
```

CHIAMATA: stampaArray3(arr)

```
void stampaArray4 (int * v) {
  int i;

 for (i=0; i<N; i++)
   printf("componente %d = %d\n", i, v[i]);
return;
}
```

CHIAMATA: stampaArray4(arr)

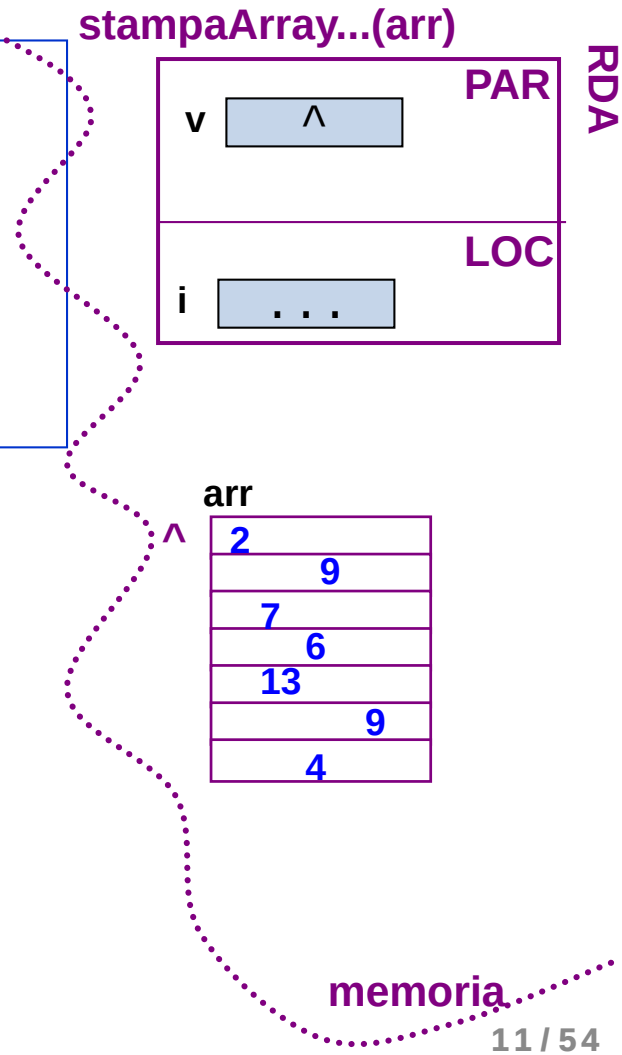


Scansione array in vari modi – 4/4 –

stampaArray2 riceve un parametro di tipo array e lo usa come se fosse un puntatore
stampaArray4 riceve un parametro di tipo puntatore (riferito ad un array) e lo usa come se fosse un nome di array

```
void stampaArray1 (int v[N]) { … }

void stampaArray2 (int v[N]) { … }

void stampaArray3 (int *v) { … }

void stampaArray4 (int *v) { … }
```

CHIAMATA: stampaArray#(arr)



NB

Le chiamate funzione(arr) sembrano tutte uguali: prendono il medesimo parametro:

A livello di tipi nei parametri formali delle funzioni

int * v e int v[] sono equivalenti

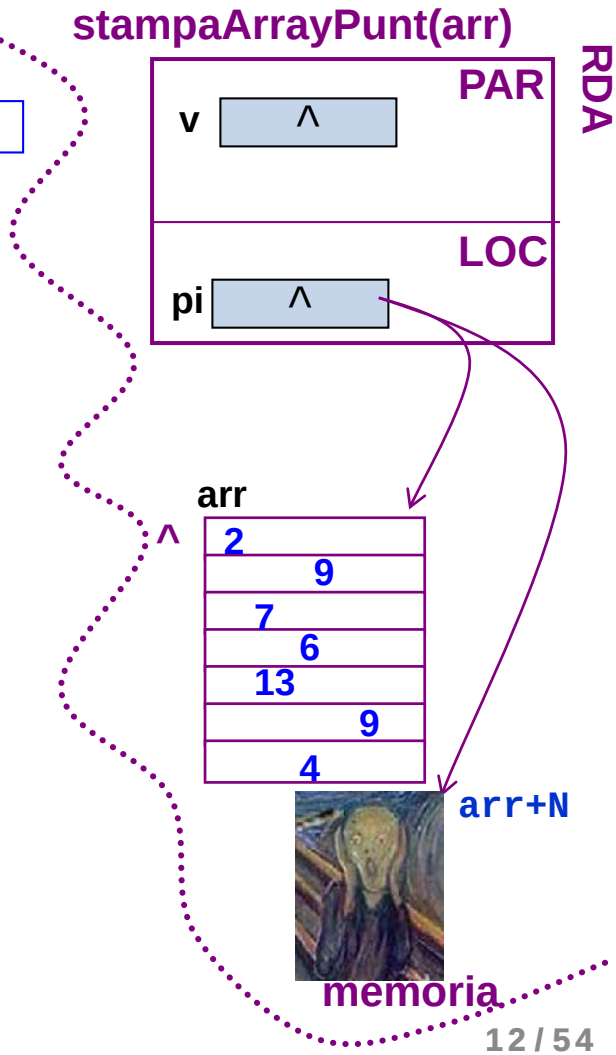
Scansione array in vari modi – scansione array mediante puntatore

come sopra: ora pero` la funzione usa un puntatore per scandire l'array (e non piu` l'intero i)

```
#define N 7
...
void stampaArrayPunt (int *v) {
  int *pi;

 for (pi=  ; pi<      ; pi++)
   printf(" - %d\n", *pi);
return;
}


int main() {
  int arr[N] = {...};
...
stampaArrayPunt(arr);
...
...
}
```

pi punta dopo l'ultima locazione

pi punta sulla prima locazione



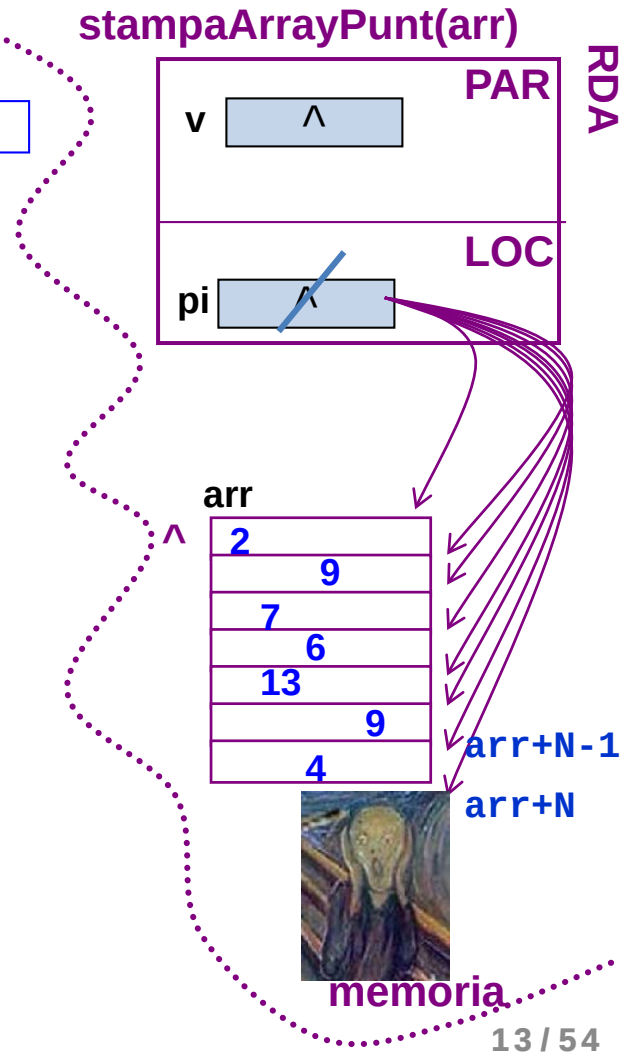
Scansione array in vari modi – scansione array mediante puntatore

come sopra: ora pero` la funzione usa un puntatore per scandire l'array (e non piu` l'intero i)

```
#define N 7
...
void stampaArrayPunt (int *v) {
  int *pi;

 for (pi= v; pi< v+N  ; pi++)
   printf(" - %d\n", *pi);
return;
}


int main() {
  int arr[N] = {...};
...
stampaArrayPunt(arr);
...
...
}
```

pi punta dopo l'ultima locazione

pi punta sulla prima locazione



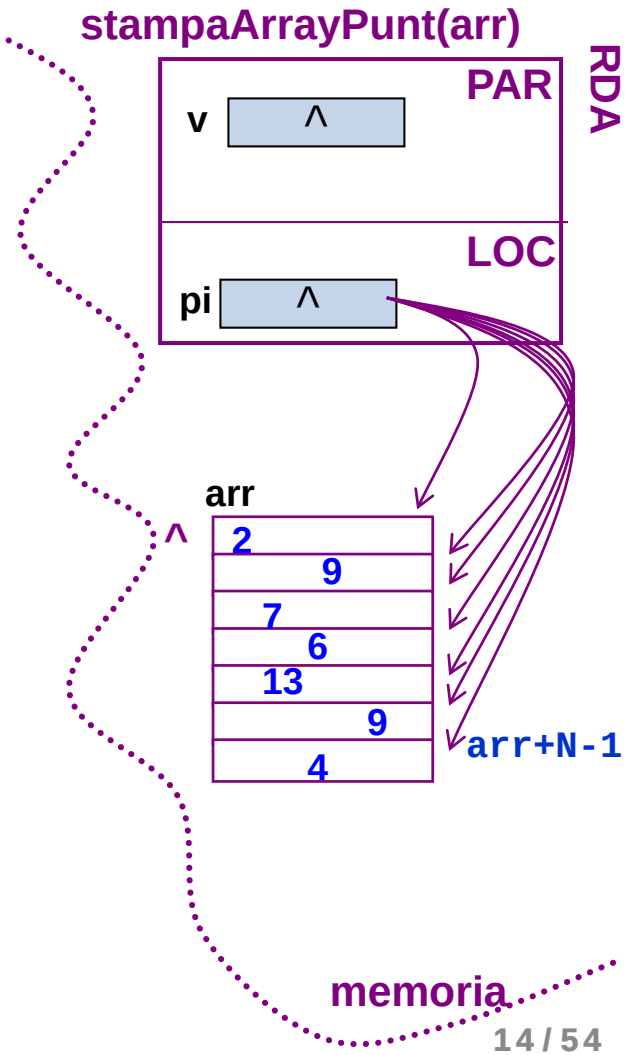
Scansione array in vari modi – scansione array mediante puntatore 3

come sopra: ora pero` la funzione usa un puntatore per scandire l'array (e non piu` l'intero i)

```
#define N 7
...
void stampaArrayPunt (int *v) {
  int *pi;

 for (pi=v; pi< v+N; pi++)
   printf(" - %d\n", *pi);
return;
}


int main() {
  int arr[N] = {...};
...
stampaArrayPunt(arr);
...
...
}
```

≡&v[0]

≡&v[N]



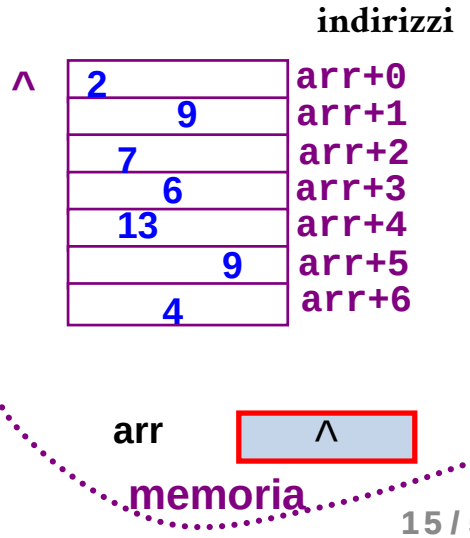
Scansione array mediante puntatore: Es. "somma dei primi m"

esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**m**)

restituisca la somma dei primi m elementi di v (o tutti se m>=n)

```
/* 1a fase: ambiente di calcolo */
#define DIM 7
                (**)
int main() {
  int arr[DIM]={2,9,...,4};
  int quanti, sommaElem;

 quanti = 4; /* scanf("%d", &quanti); */
 sommaElem = sommaPrimi(arr, DIM, quanti);
 printf("... %d ...", sommaElem);

return 0;
}
```



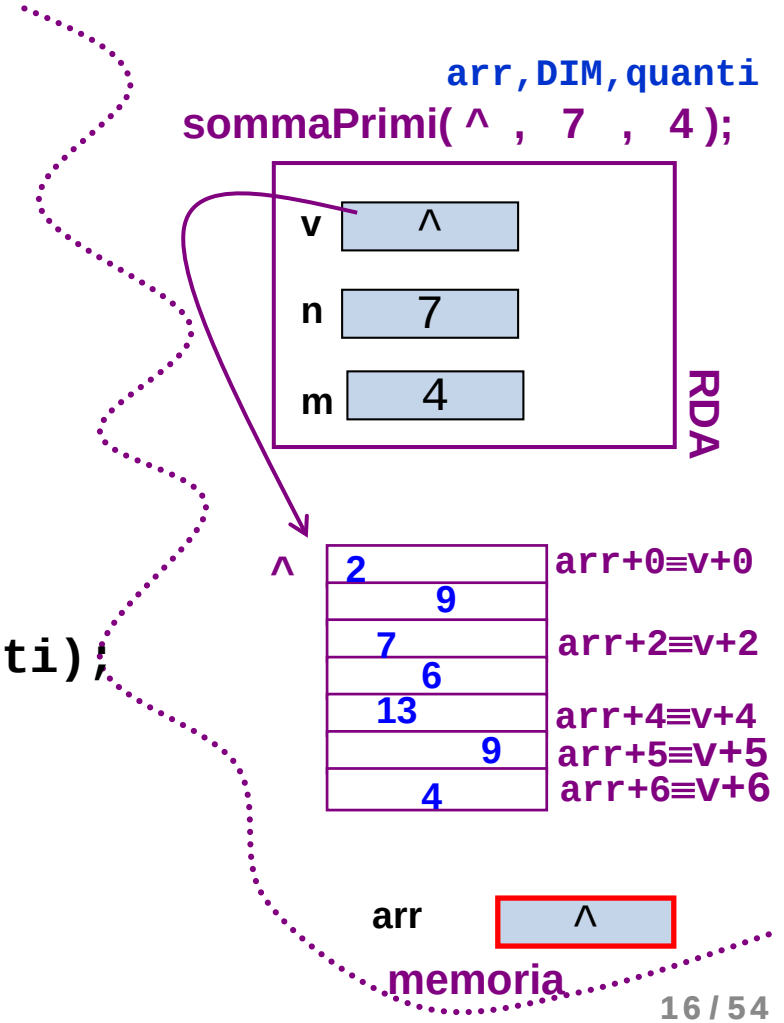
Es. "somma dei primi m" (1/4)

esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**m**)

restituisca la somma dei primi m elementi di v (o tutti se m>=n)

```
/* 1a fase: ambiente di calcolo */
#define DIM 7
                (**)
int main() {
  int arr[DIM]={2,9,...,4};
  int quanti, sommaElem;

 quanti = 4; /* scanf("%d", &quanti); */
 sommaElem = sommaPrimi(arr, DIM, quanti);
 printf("... %d ...", sommaElem);

return 0;
}
```



Es. "somma dei primi m" (2/4)

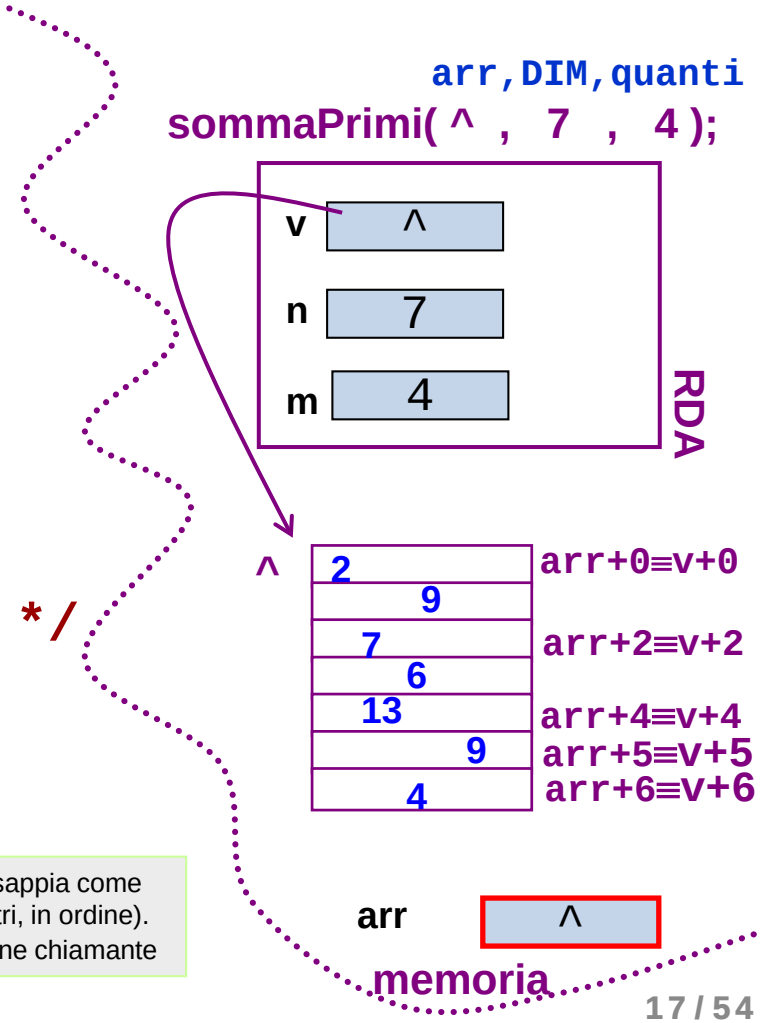
esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**m**)

restituisca la somma dei primi m elementi di v (o tutti se m>=n)

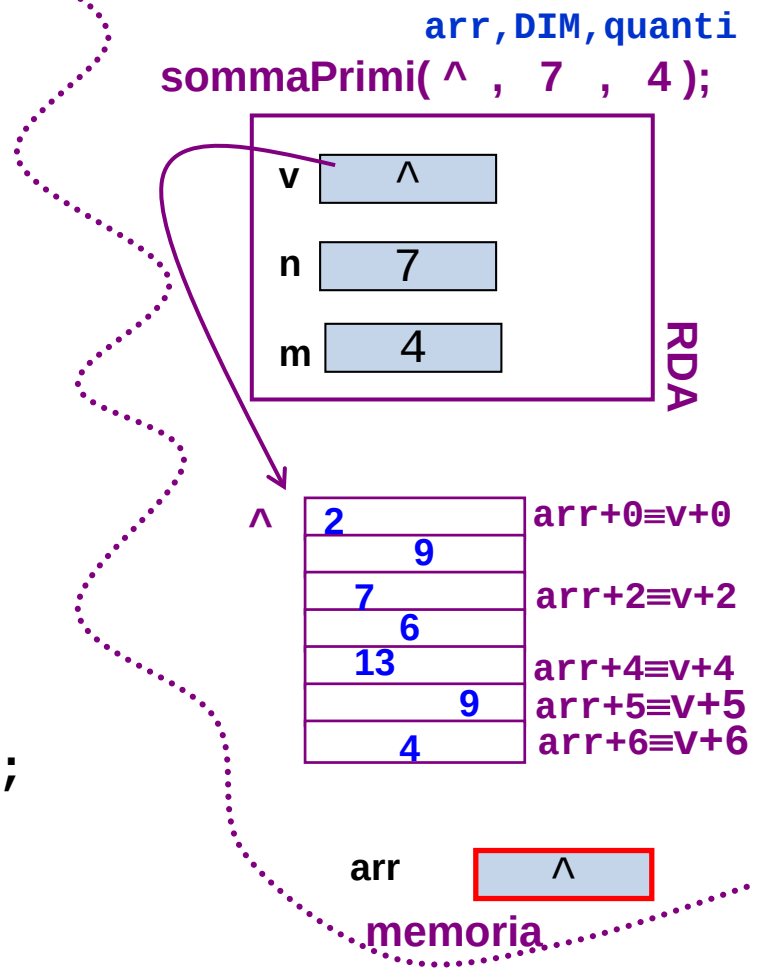
```
/* 1a fase: ambiente di calcolo */
#define DIM 7
                    (**)
int main() {
  int arr[DIM]={...};
  int quanti, sommaElem;

 quanti = 4;
 sommaElem = sommaPrimi(arr, DIM, quanti);
 printf("... %d ...", sommaElem);

return 0;
}
```

```
(**)
/* 2a fase: PROTOTIPO (dichiarazione) */
int sommaPrimi (int *, int, int);
```

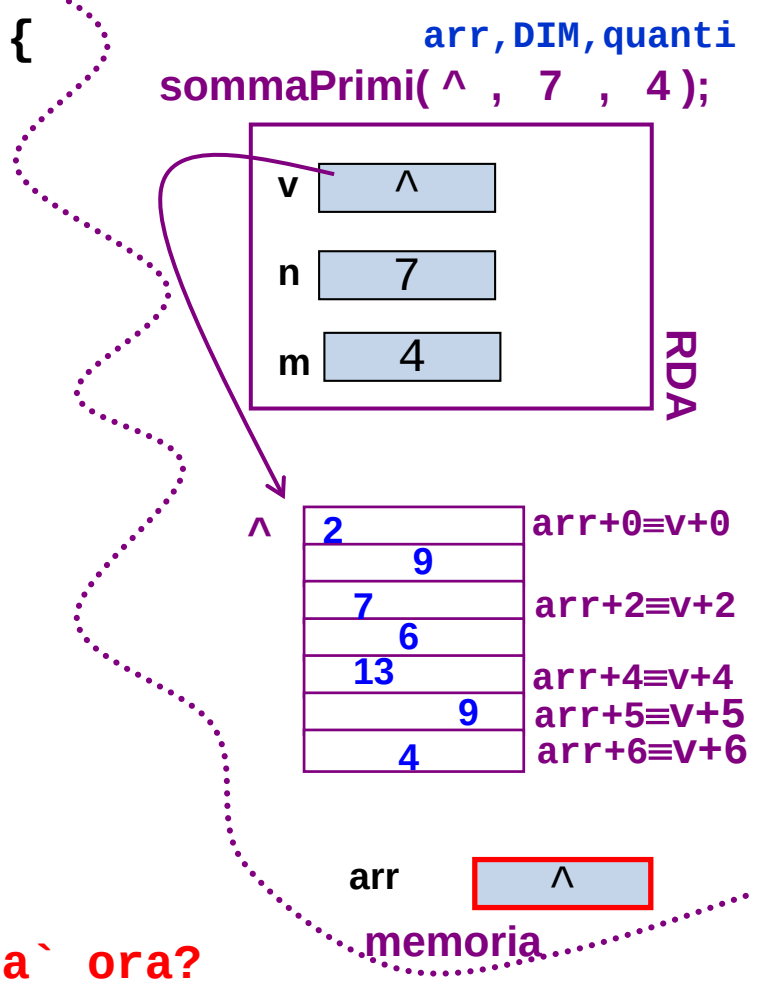


Il prototipo serve a DICHIARARE la funzione, in modo che il codice successivo “la conosca” (sappia come chiamarla): il prototipo dice 1) il nome della funzione; 2) il tipo del risultato; 3) i tipi dei parametri, in ordine). Se la funzione e` dichiarata, la sua effettiva DEFINIZIONE puo` essere ritardata oltre la funzione chiamante

```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {

alg. provvisorio …

scandire gli elementi dell'array,

ma solo quelli dal primo all'm-esimo,

accumulandoli in una
variabile "accumulatore" somma

e poi restituire somma ... return somma;
}
```



```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {

alg. provvisorio …

   ripeti, per i che va da 0 a m-1
      somma += elemento di indice i

return somma;
}
```



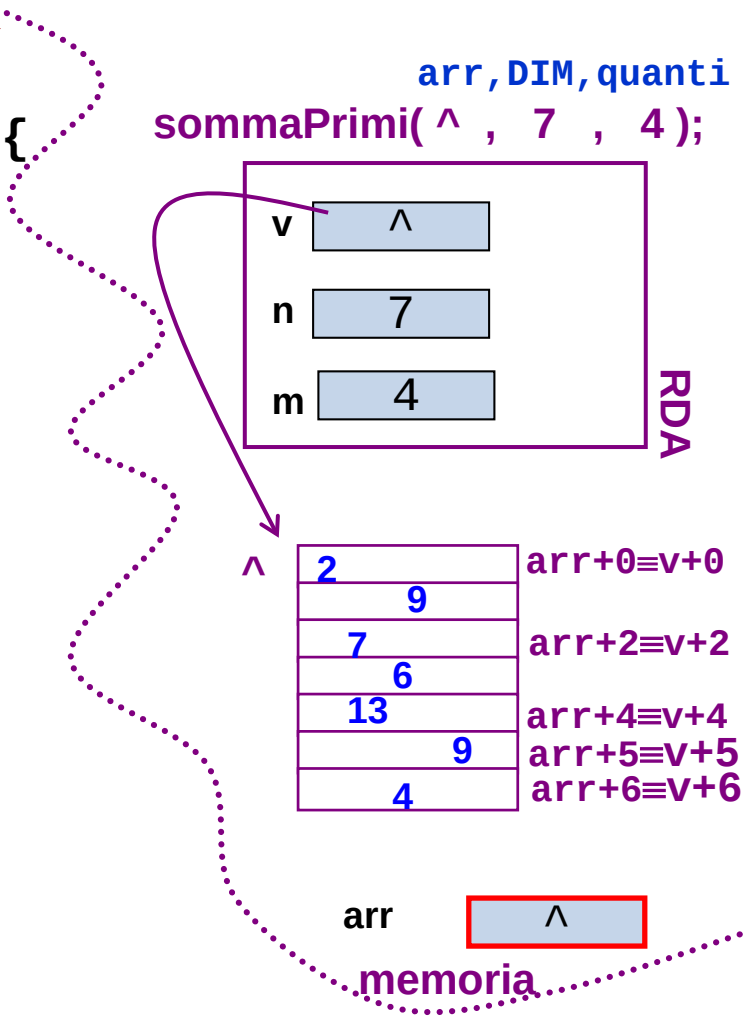
☺ qualche osservazione gia` ora?

☺ scriverlo completo (passo 0, passo 1, return

```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {


alg. provvisorio …
0) usa i, somma
   sono interi, con somma da init a 0
   (int i, somma=0;)
1) ripeti, per i che va da 0 a m-1
   1.1) somma += elemento di indice i


return somma;
}
```

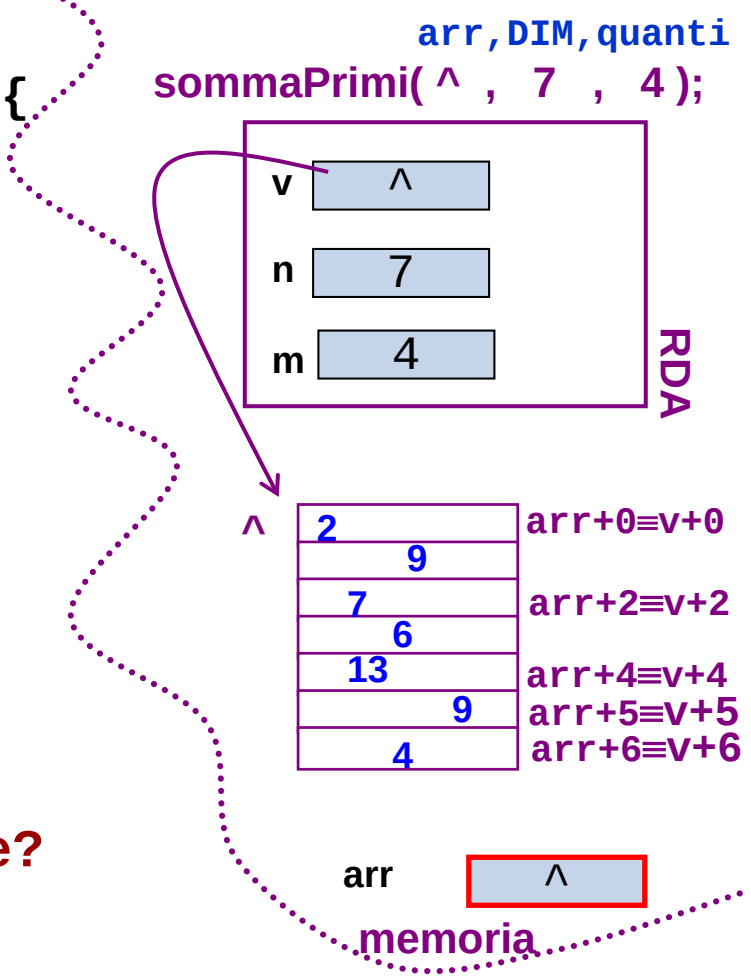
ok, somma va inizializzata. Giusto.

☺ qualcosa da aggiustare ci sarebbe ancora ... forse ...



Es. "somma dei primi m" (4/4) NB implementazione, ma ...

```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {
  int i, somma=0;

 for (i=0; i<m; i++)
    somma += *(v+i);

return somma;
}
```



4a fase? Test quali test fare?

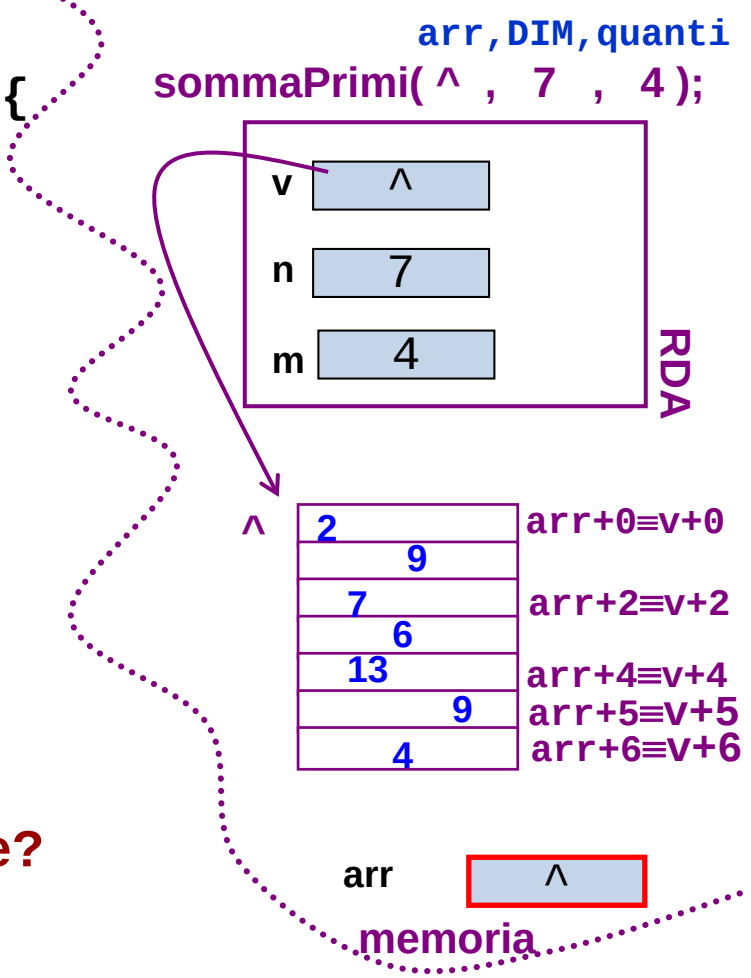
☺

progettare qualche test di prova, utile a capirese questa funzione funge ... un test consiste nel lanciare la funzione con "certi dati di input" – in questo caso con certi valori per i parametri attuali ... e vedere cosa succede

(n==0 ?! n<0?!! Buona idea, ma lasciamoli stare qui)

Es. "somma dei primi m" (4/4) NB implementazione: testiamola

```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {
  int i, somma=0;

 for (i=0; i<m; i++)
    somma += *(v+i);

return somma;
}
```

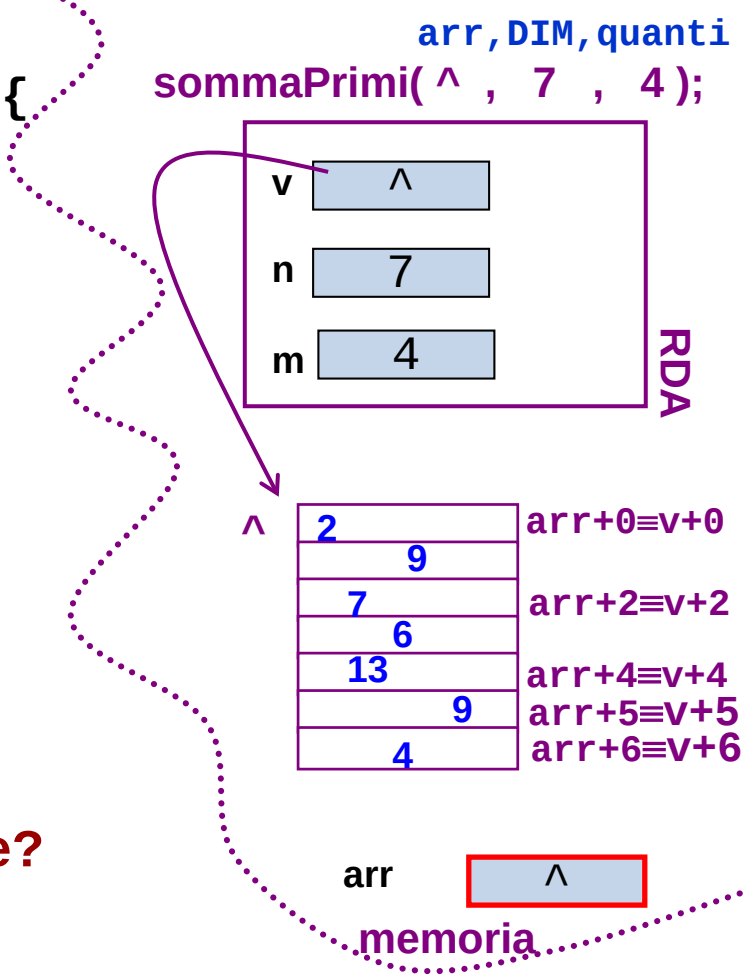


4a fase?	Test	quali test fare?
m < n		☺ funge?
m == n		☺ funge?

(n==0 ?! n<0?!! lasciamoli stare)

Es. "somma dei primi m" (4/4) NB implementazione: altri test ...

```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {
  int i, somma=0;

 for (i=0; i<m; i++)
    somma += *(v+i);

return somma;
}
```



4a fase?	Test	quali test fare?
	m > n	☺ funge?
	m < 0	☺ funge?

(n==0 ?! n<0?!! lasciamoli stare)

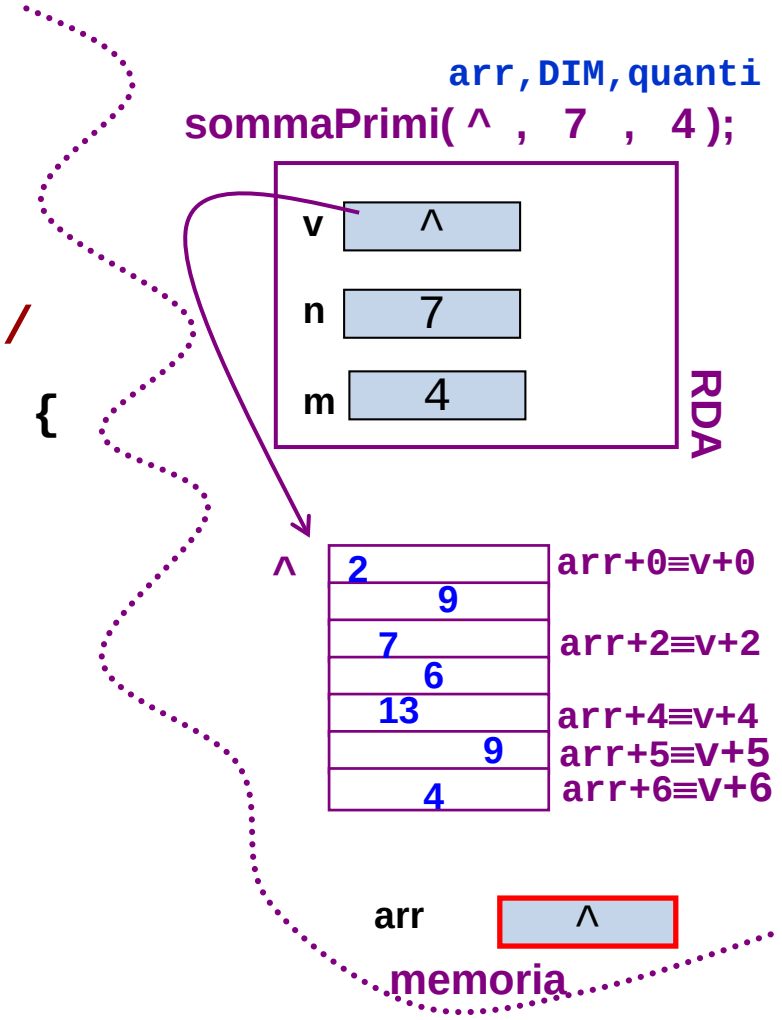
```
/* 3a fase: definizione funzione */
int sommaPrimi (int * v, int n, int m) {
  int i, somma=0;
 if (m>n) m=n;
 for (i=0; i<m; i++)
    somma += *(v+i);

return somma;
}
```

4a fase? Test

m < n / m == n

m > n / m < 0

(n==0 ?! n<0?!! lasciamoli stare)



Ricerca in array, mediante puntatori - TrovaElemento() -

esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)

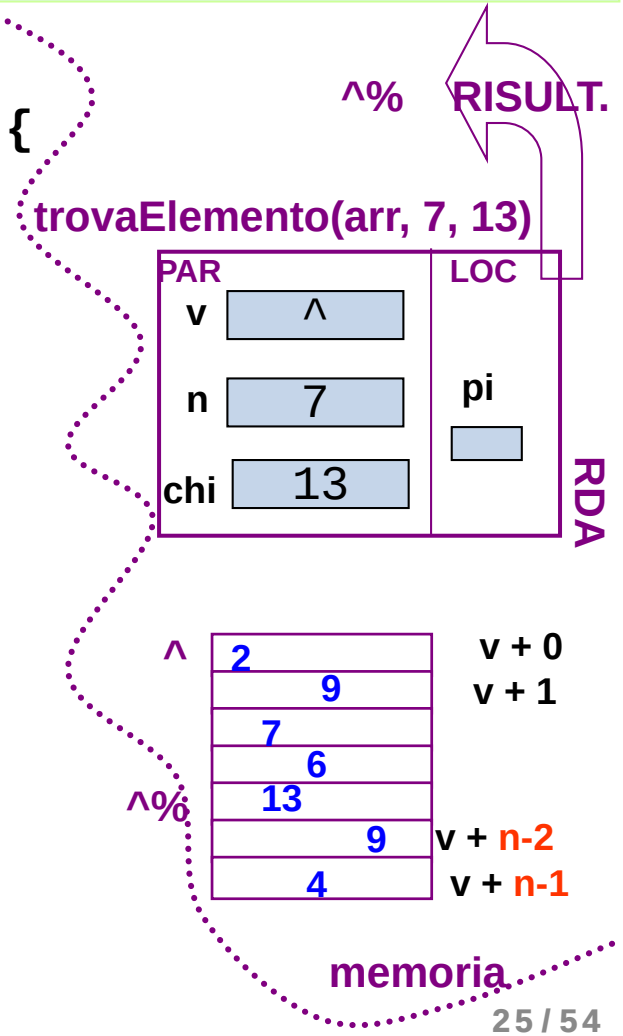
restituisca il puntatore al 1º elemento di v che e` uguale a **chi**

```
/* definizione della funzione */
int * trovaElemento(int *v, int n, int chi) {
  int *pi = v; /* NB init in definizione */
```

scansione e controllo di ogni elemento dell'array

- quando pi == v+n (arr+n), vuol dire che poco prima pi puntava sull'ultimo elem. e ora pi punta fuori dell'array;
- se si scopre che chi==*pi, pi e` l'indirizzo cercato e si esce restituendo pi;
- se nessun elemento e` == chi, arrivati fuori dell'array si rest. NULL

```
return NULL;
}
```



Ricerca in array, mediante puntatori - TrovaElemento() -

esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)

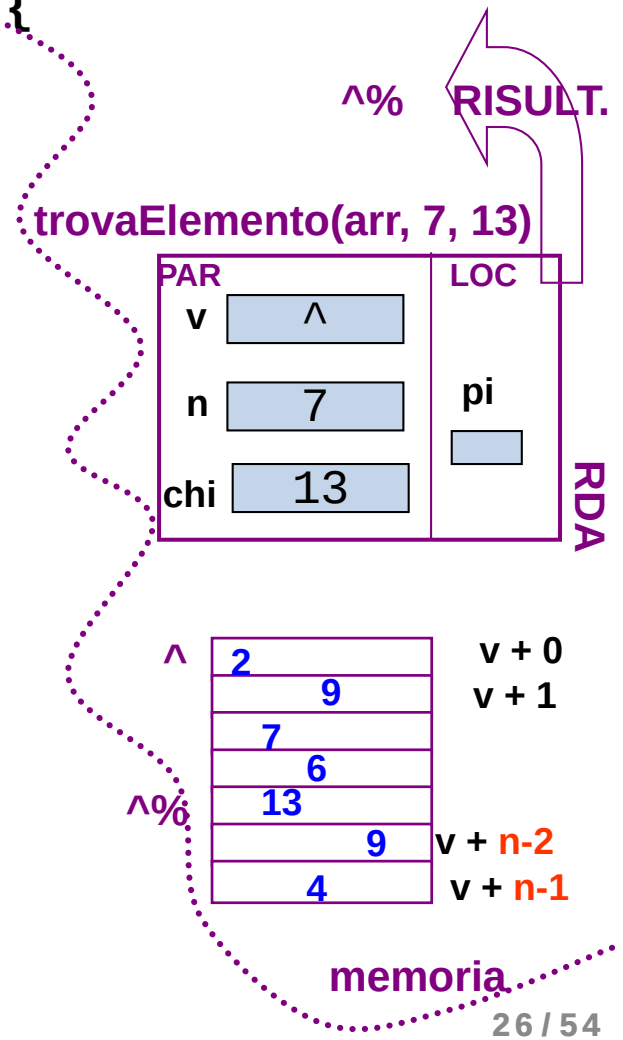
restituisca il puntatore al 1º elemento di v che e` uguale a **chi**

```
/* definizione della funzione */
int * trovaElemento(int *v, int n, int chi) {
  int *pi = v; /* NB init in definizione */
```

scansione e controllo di ogni elemento dell'array

```
 while (pi != v+n) {
    if (*pi == chi)
      return pi;

    pi++;
 }
return NULL;
}
```

- quando pi == v+n (arr+n), vuol dire che poco prima pi puntava sull'ultimo elem. e ora pi punta fuori dell'array;
- se si scopre che chi==*pi, pi e` l'indirizzo cercato e si esce restituendo pi;
- se nessun elemento e` == chi, arrivati fuori dell'array si rest. NULL



Test

- chi e` in array (in mezzo) ?
- chi non c'e` ?
- chi c'e` al primo posto ?
- chi c'e` all'ultimo posto ?

Ricerca in array, mediante puntatori - TrovaUltima() - 1/2

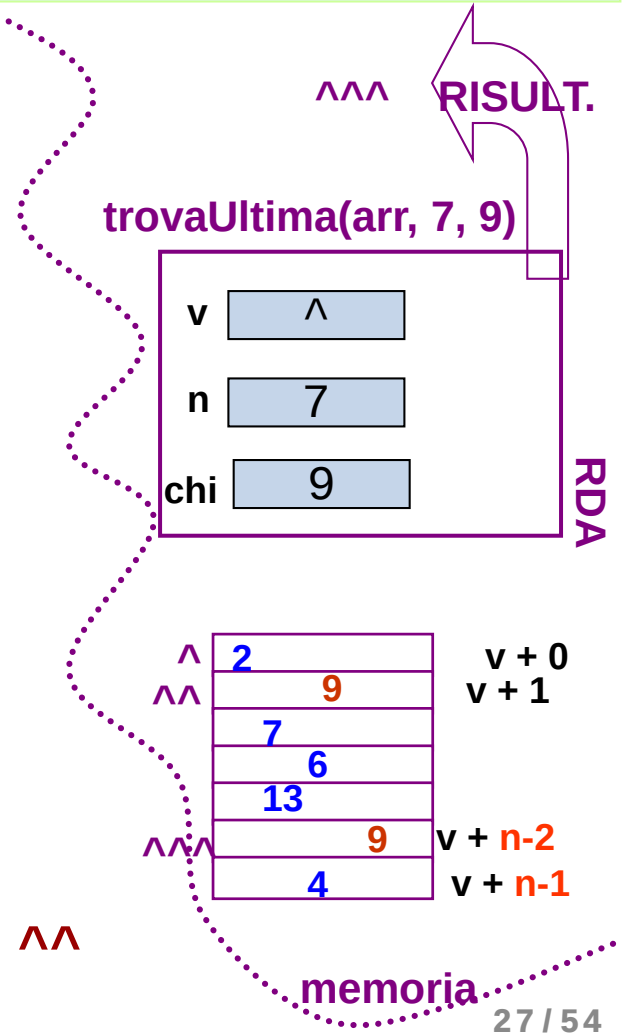
esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di **v**, e un intero (**chi**)

restituisca il puntatore alla **ultima occorrenza** di **chi** in **v** (opp. NULL)

```
/* 1a fase: ambiente di calcolo */
#define DIM 7

int * trovaUltima (int *, int, int)

int main() {
  int daTrovare=9, arr[DIM];
  int *pRis; /* risultato */

 /* … lettura array … */

pRis = trovaUltima(arr, DIM, daTrovare);

if (pRis != NULL)
  printf("trovato %d\n", *pRis);
else  printf("tsk ... tsk ...\n");
return 0;
}
```

risultato ^^^ e non ^^



Ricerca in array, mediante puntatori - TrovaUltima() - 2/2

esercizio ultima occorrenza

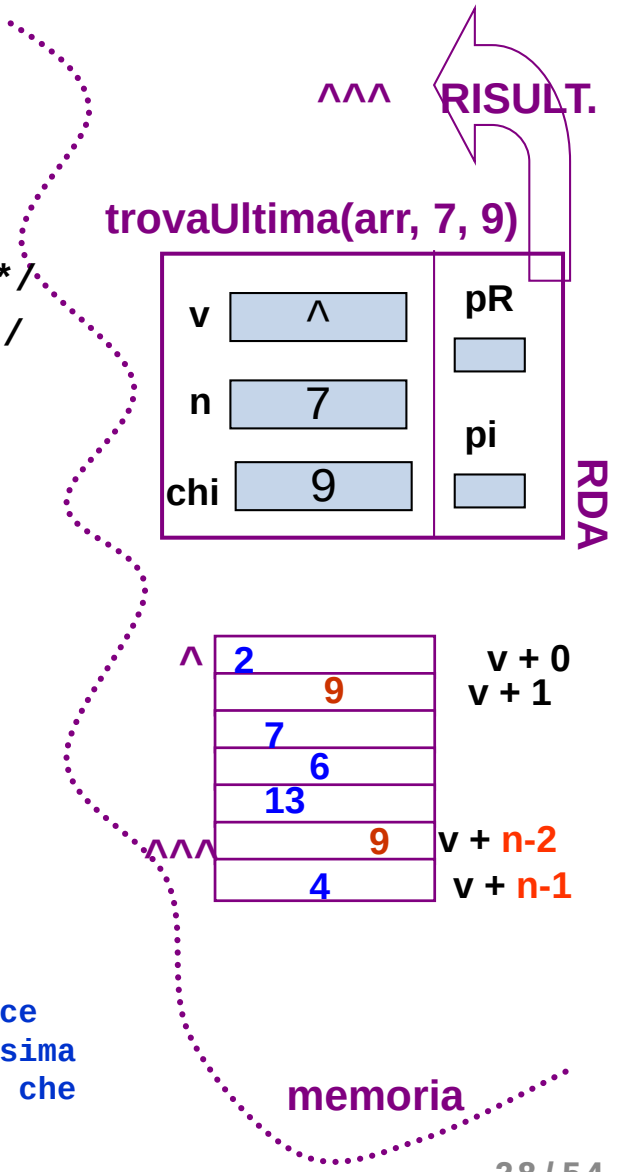
```
/* definizione funzione */
int * trovaUltima (int *v, int n, int chi)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL)   */
        *pi = v;       /* per scandire array       */
```

☺ alg?

un po' diverso dal caso precedente: qui

quando troviamo "chi" (cioe` quando *pi == chi) dobbiamo

- segnarci che all'indirizzo pi c'e` chi;
- ma poi continuare a cercare chi (e quindi non si finisce qui, ma si incrementa comunque pi, in modo che alla prossima scansione si controllerà il prossimo elemento ... a meno che non sia finito l'array (pi==v+n)



Ricerca in array, mediante puntatori - TrovaUltima() - 2/2

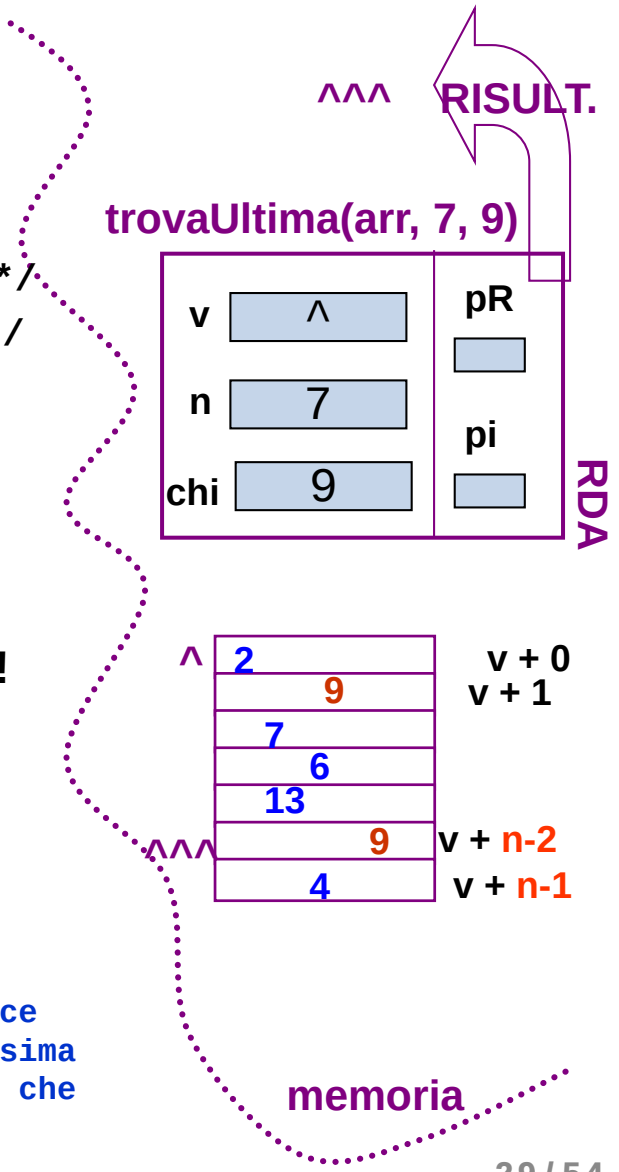
esercizio ultima occorrenza

```
/* definizione funzione */
int * trovaUltima (int *v, int n, int chi)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL)   */
        *pi = v;       /* per scandire array       */

 0) *pR per il risultato; *pi per scandire
    le locazioni dell'array

 1) mentre pi < v+n
   1.1) se *pi e` uguale a chi: TROVATO UNO!
        pR = pi
   1.2) avanza pi sul prossimo elemento
```

un po' diverso dal caso precedente: qui

quando troviamo "chi" (cioe` quando *pi == chi) dobbiamo

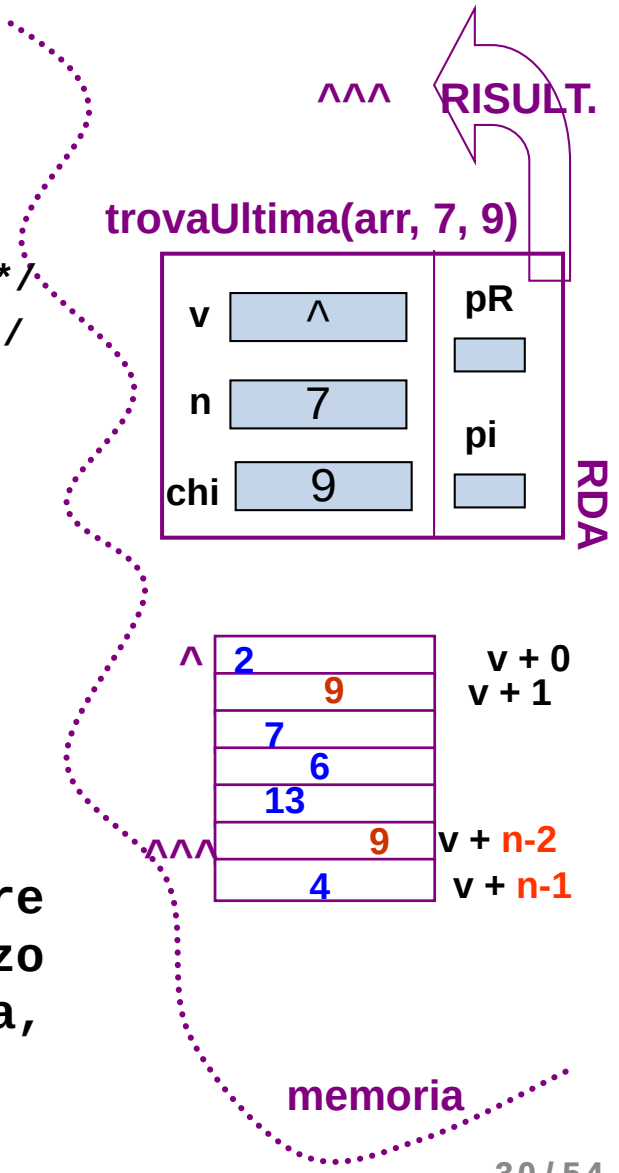
- segnarci che all'indirizzo pi c'e` chi;
- ma poi continuare a cercare chi (e quindi non si finisce qui, ma si incrementa comunque pi, in modo che alla prossima scansione si controllerà il prossimo elemento ... a meno che non sia finito l'array (pi==v+n)



Ricerca in array, mediante puntatori - TrovaUltima() - 2/2

esercizio ultima occorrenza

```
/* definizione funzione */
int * trovaUltima (int *v, int n, int chi)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL)   */
        *pi = v;       /* per scandire array       */

 0) *pR per il risultato; *pi per scandire
    le locazioni dell'array

 1) mentre pi < v+n
   1.1) se *pi e` uguale a chi TROVATO UNO!
        pR = pi
   1.2) avanza pi sul prossimo elemento

 2) ciclo finito. In *pR c'e` l'ultimo valore
 che gli abbiamo assegnato, cioe` l'indirizzo
 dove abbiamo trovato chi per l'ultima volta,
 cioe` il risultato!
```



Ricerca in array, mediante puntatori - TrovaUltima() - 2/2

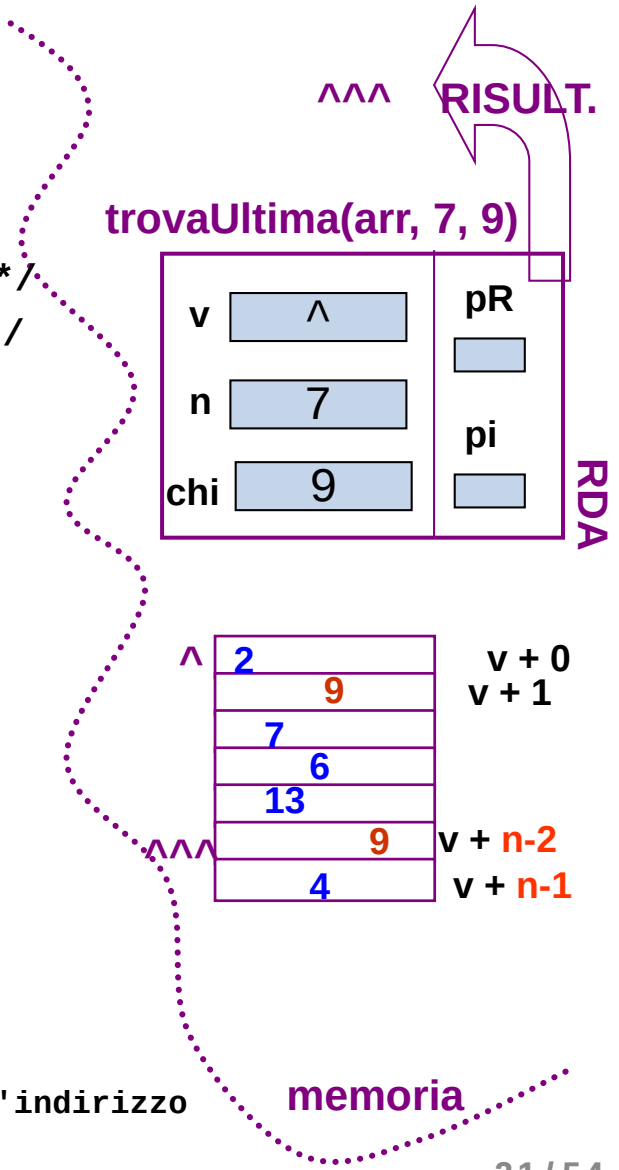
esercizio ultima occorrenza

```
/* definizione funzione */
int * trovaUltima (int *v, int n, int chi)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL)   */
        *pi = v;       /* per scandire array       */
```

☺ codice?

```
return pR;
}
```

```
0) *pR per il risultat; *pi per scandire le locazioni dell'array
1) mentre pi < v+n
  1.1) se *pi e` uguale a chi TROVATO UNO!
       pR = pi
  1.2) avanza pi sul prossimo elemento
2) in *pR c'e` l'ultimo valore che gli abbiamo assegnato, cioe` l'indirizzo
dove abbiamo trovato chi per l'ultima volta, cioe` il risultato!
```

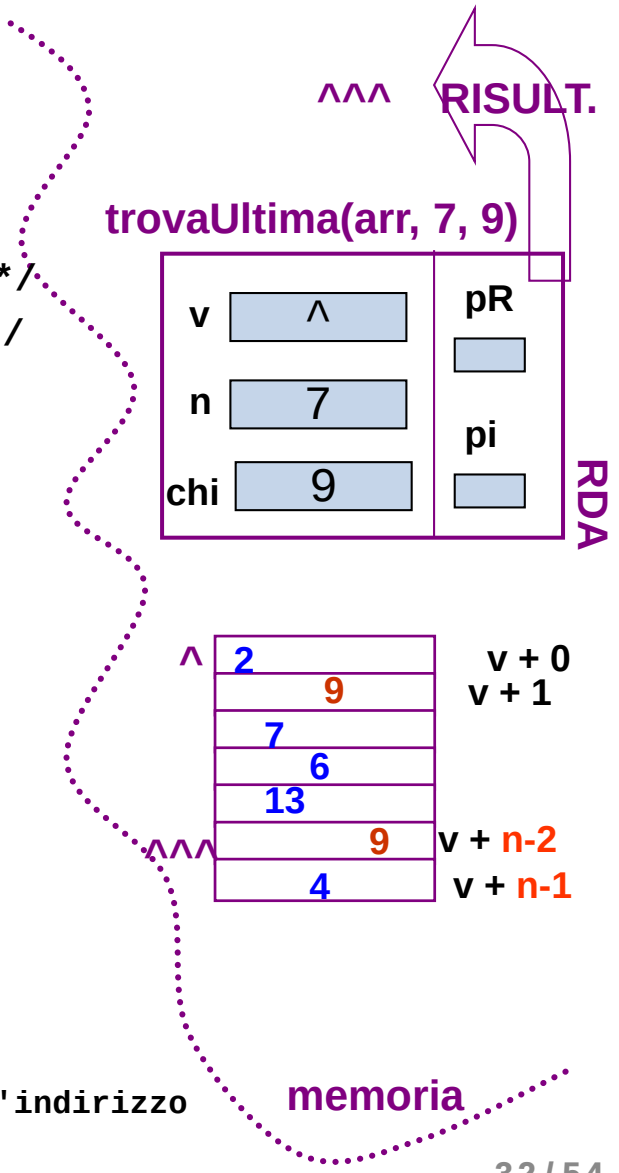


Ricerca in array, mediante puntatori - TrovaUltima() - 2/2

esercizio ultima occorrenza

```
/* definizione funzione */
int * trovaUltima (int *v, int n, int chi)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL)   */
        *pi = v;       /* per scandire array       */

 while( pi < v+n) {
   if (*pi == chi)
      pR = pi;
   pi++;
 }
return pR;
}
```

```
0) *pR per il risultat; *pi per scandire le locazioni dell'array
1) mentre pi < v+n
  1.1) se *pi e` uguale a chi TROVATO UNO!
       pR = pi
  1.2) avanza pi sul prossimo elemento
2) in *pR c'e` l'ultimo valore che gli abbiamo assegnato, cioe` l'indirizzo
dove abbiamo trovato chi per l'ultima volta, cioe` il risultato!
```



Ricerca in array, mediante puntatori - TrovaUltima() - 2/2

esercizio ultima occorrenza

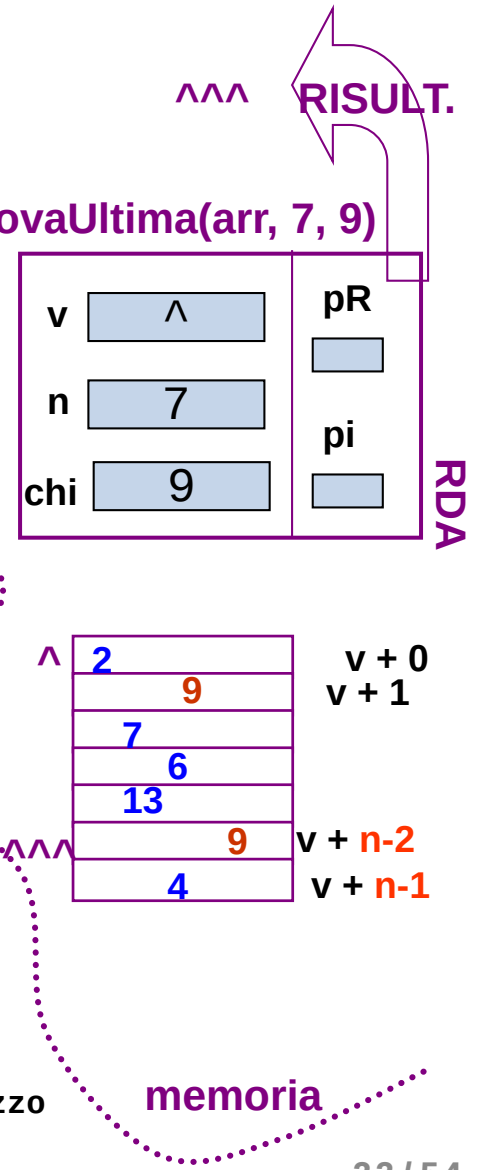
```
/* definizione funzione */
int * trovaUltima (int *v, int n, int chi)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL)   */
        *pi = v;       /* per scandire array       */

 while( pi < v+n) {
   if (*pi == chi)
      pR = pi;
   pi++;
 }
return pR;
}
```

0) *pR per il risu
1) mentre pi < v+n
  1.1) se *pi e` u
       pR = pi
  1.2) avanza pi su
2) in *pR c'e` l'u
dove abbiamo trovat

izzo

Test

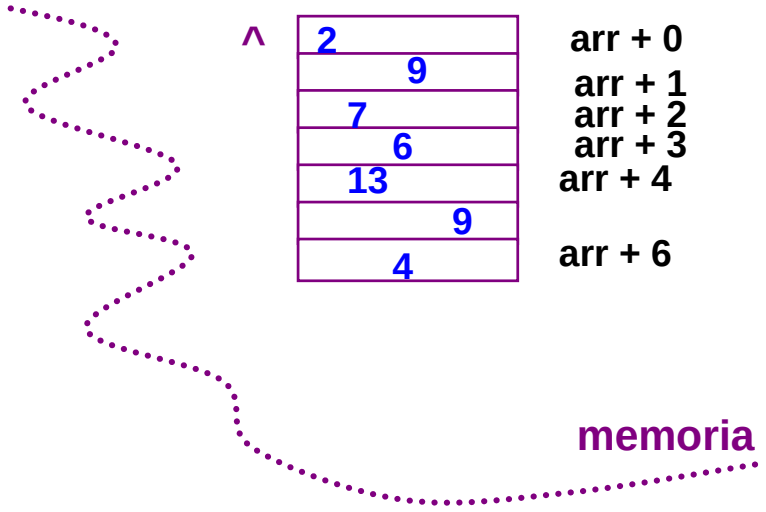
- chi e` in array (due volte) ?
- chi e` in array (una volta) ?
- chi non c'e` ?
- chi e` in array (una volta, al primo posto) ?
- ultima occorrenza all'ultimo posto ?



Riuso di un modulo e gioco di puntatori

esercizio **dato** un array di 7 elementi interi, **stampare** la somma
A) dei primi 5 **B)** di quelli dal 2º al 5º

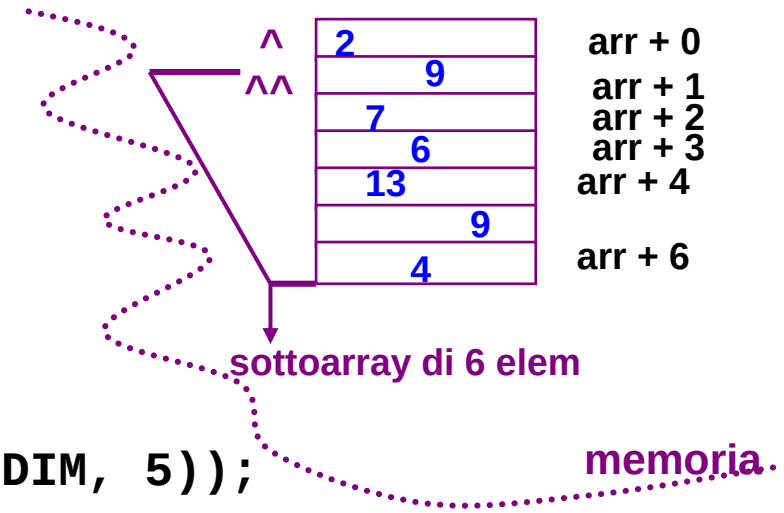
```
/* 1a fase: ambiente di calcolo */
#define DIM 7
int sommaPrimi(int *, int, int);

int main() {
  int arr[DIM]={...};

/* A) */
 printf("somma primi 5: %d\n",      ☺
```



Riuso di un modulo e gioco di puntatori

esercizio **dato** un array di 7 elementi interi, **stampare** la somma
A) dei primi 5 **B)** di quelli dal 2º al 5º

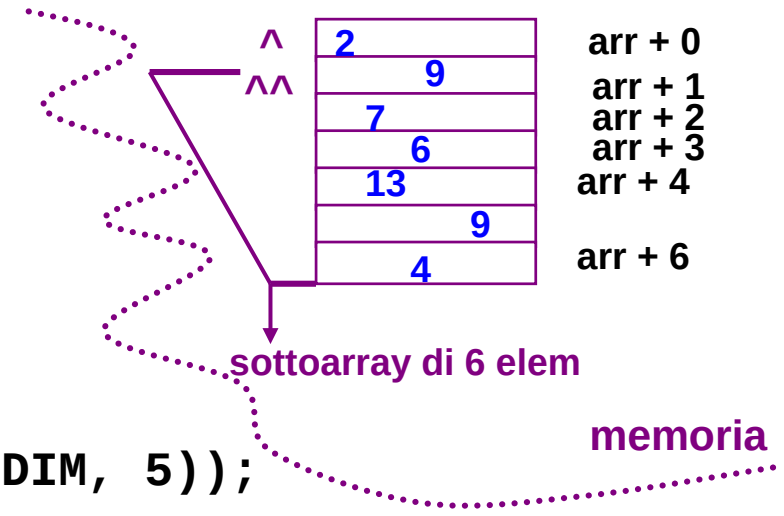
```
/* 1a fase: ambiente di calcolo */
#define DIM 7
int sommaPrimi(int *, int, int);

int main() {
  int arr[DIM]={...};

/* A) */
 printf("somma primi 5: %d\n",
                        sommaPrimi(arr, DIM, 5));
/* B)                  nell'array di indirizzo iniziale arr+1
                       e lunghezza 6 (DIM-1)
                       sommare i primi 4 (5-1)
*/
printf("somma da sec a quinto: %d\n",          ☺
```



Riuso di un modulo e gioco di puntatori

esercizio **dato** un array di 7 elementi interi, **stampare** la somma
A) dei primi 5 **B)** di quelli dal 2º al 5º

```
/* 1a fase: ambiente di calcolo */
#define DIM 7
int sommaPrimi(int *, int, int);

int main() {
  int arr[DIM]={...};

/* A) */
 printf("somma primi 5: %d\n",
                        sommaPrimi(arr, DIM, 5));
/* B)                  nell'array di indirizzo iniziale arr+1
                       e lunghezza 6 (DIM-1)
                       sommare i primi 4 (5-1)
*/
printf("somma da sec a quinto: %d\n",
                                    sommaPrimi(arr+1, DIM-1, 4));
return 0;
}
```



PARAMETRI DI OUTPUT - 1/7 -

esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)

restituisca 1) puntatore alla **ultima occorrenza** di chi in v (opp. NULL)

due cose 2) numero di occorrenze di chi in v

(restituisce 2 risultati ?!)

un risultato (il primo) viene restituito dalla chiamata tramite return (come nell'esercizio precedente);

l'altro risultato viene gestito come parametro di output:

si tratta di un parametro della funzione (chiamata),

che viene collegato ad una variabile della funzione chiamante, in modo che la funzione chiamata possa eseguire su quella variabile un "side effect" (effetto collaterale) durante la sua esecuzione.

... ma arriviamoci ...

PARAMETRI DI OUTPUT - 2/7 -

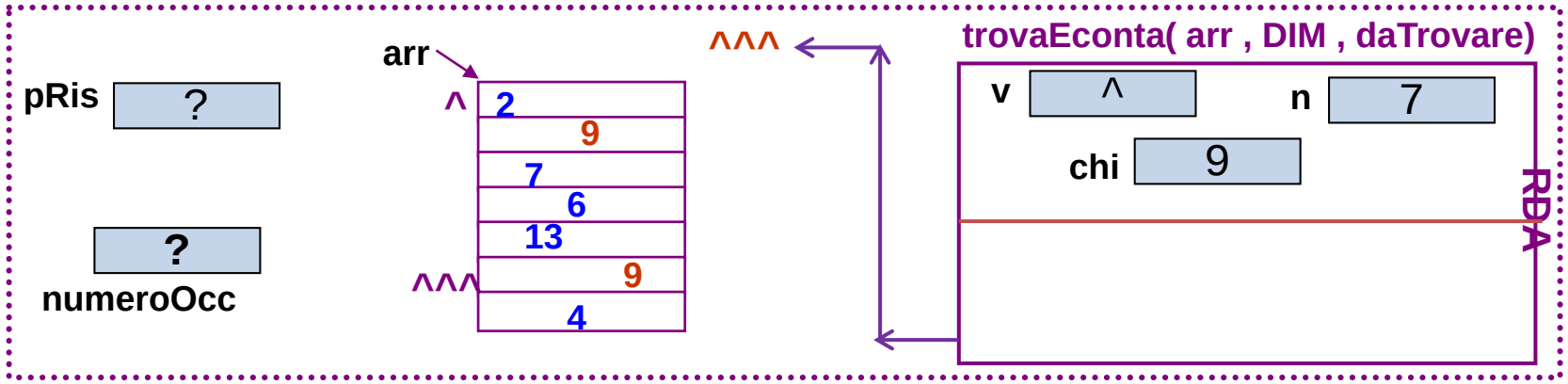
esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)

restituisca - puntatore alla **ultima occorrenza** di chi in v (opp. NULL)
- numero di occorrenze di chi in v

Primo tentativo ... come va con il primo risultato?

```
int main() {        int   daTrovare = 9,      arr[DIM],
                          numeroOcc,          *pRis;
...
  pRis = trovaEconta(arr, DIM, daTrovare);  /* puo` modificare
                                               numeroOcc???    */
...
```



PARAMETRI DI OUTPUT - 3/7 -

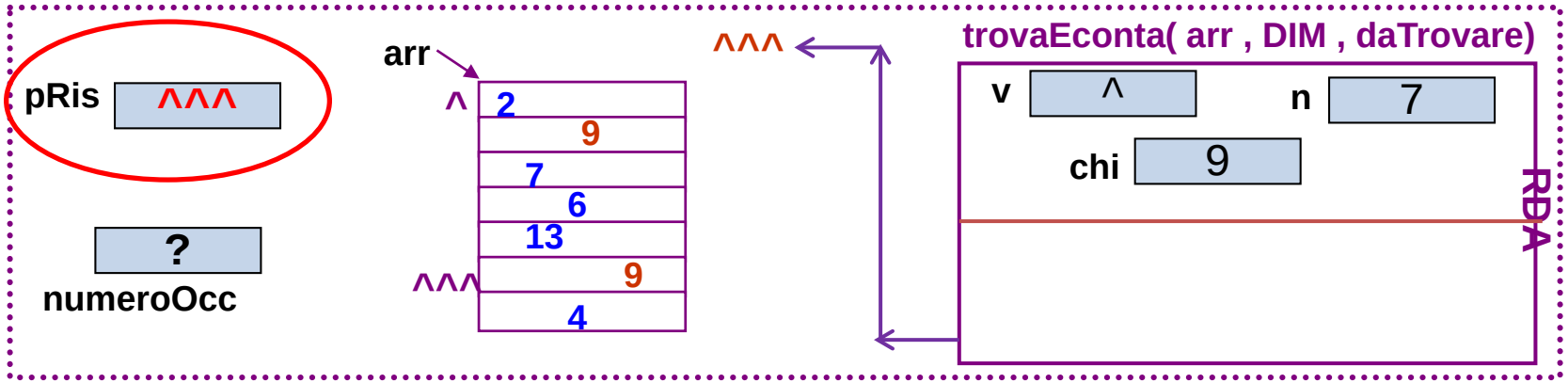
esercizio funzione che

ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)

restituisca - puntatore alla **ultima occorrenza** di chi in v (opp. NULL)
- numero di occorrenze di chi in v

Primo risultato: **ovviamente OK ... what about the second?** 😐

```
int main() {          int   daTrovare = 9,      arr[DIM],
                            numeroOcc,          *pRis;
...
  pRis = trovaEconta(arr, DIM, daTrovare);  /* puo` modificare
                                               numeroOcc???    */
...
```



PARAMETRI DI OUTPUT - 4/7 -

esercizio funzione che
ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)
restituisca - puntatore alla **ultima occorrenza** di chi in v (opp. NULL)
- numero di occorrenze di chi in v

Come va con il secondo risultato?

Mettiamo nella funzione una istruzione che modifica numeroOcc ... voila`

```
int main() {        int   daTrovare = 9,      arr[DIM],
                          numeroOcc,          *pRis;
...
  pRis = trovaEconta(arr, DIM, daTrovare);  /* puo` modificare
                                               numeroOcc???    */
...
```

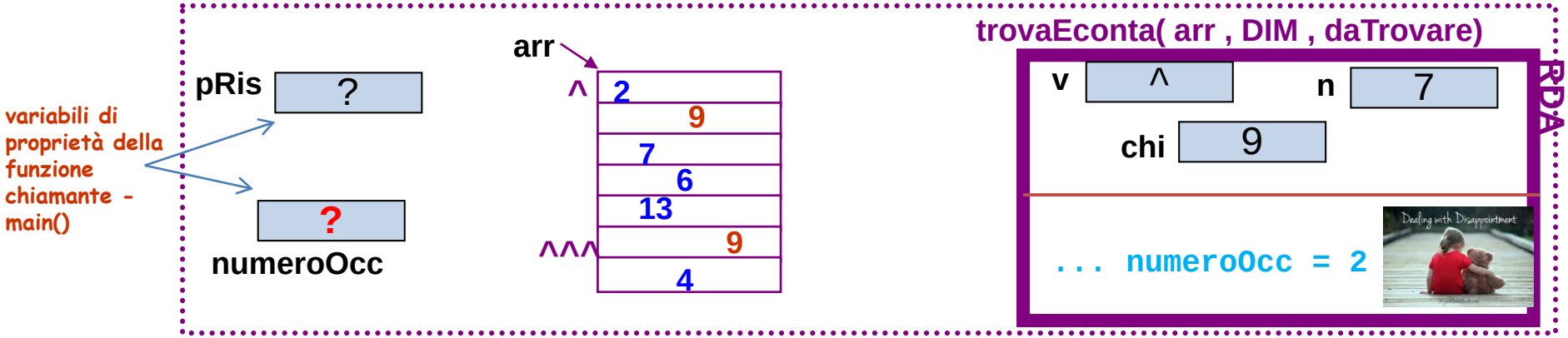
PARAMETRI DI OUTPUT - 4/7 -

esercizio funzione che
ricevendo un array (**v**) di interi, la dimensione (**n**) di v, e un intero (**chi**)
restituisca - puntatore alla **ultima occorrenza** di chi in v (opp. NULL)
- numero di occorrenze di chi in v

Mettiamo nella funzione una istruzione che modifica numeroOcc ... voila`

no ... una istruzione della funzione, che modifichi numeroOcc e` illegale (la funzione non "vede" numeroOcc).

```
int main() {          int   daTrovare = 9,      arr[DIM],
                            numeroOcc,          *pRis;
...
  pRis = trovaEconta(arr, DIM, daTrovare);  /* puo` modificare
                                               numeroOcc???    */
...
```



Uso di puntatori per PARAMETRI DI OUTPUT - 1/9 -

No … cosi` non va … in qualche modo numeroOcc deve essere un parametro della funzione, o collegato ad un parametro, in modo che una istruzione della funzione lo possa modificare. Ma come?

... ecco come

FACENDO IN MODO CHE UN PARAMETRO FORMALE DELLA FUNZIONE CHIAMATA SIA ASSEGNATO CON
L'INDIRIZZO DI UNA VARIABILE APPARTENENTE ALLA FUNZIONE CHIAMANTE:

USANDO QUELL'INDIRIZZO LA FUNZIONE CHIAMATA POTRA` PRODURRE UN EFFETTO COLLATERALE
CIOE` L'ESECUZIONE DELLE ISTRUZIONI DELLA FUNZIONE COMPORTERA` UNA MODIFICA DELLA VARIABILE APPARTEMENTE ALLA FUNZIONE CHIAMANTE

QUESTA MODIFICA VIENE OTTENUTA ACCEDENDO ALLA VARIABILE MEDIANTE INDIRIZZO, USANDO L'OPERATORE DI DEREFERENZIAZIONE

Uso di puntatori per PARAMETRI DI OUTPUT – 2/9 –

La soluzione, per gestire PARAMETRI DI OUTPUT, e` nell'uso della tecnica del PASSAGGIO DI INDIRIZZO:

un escamotage per riuscire a ottenere effetti collaterali con le funzioni *C*

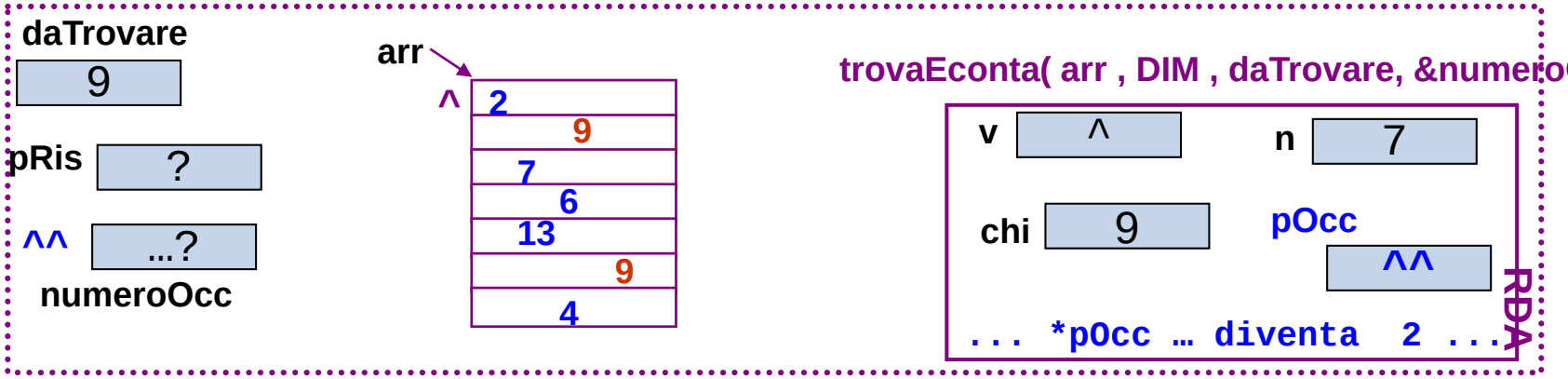
- non va passato numeroOcc (cioe` il suo valore), ma &numeroOcc,

- la chiamata sara` , nell'istruzione vista prima,

```
pRis = trovaEconta( arr , DIM , daTrovare , &numeroOcc)
```

- la definizione della funzione, con un parametro formale "di output" sara`

```
int * trovaEconta (int * v, int n, int chi, int *pOcc) {
…
}
```

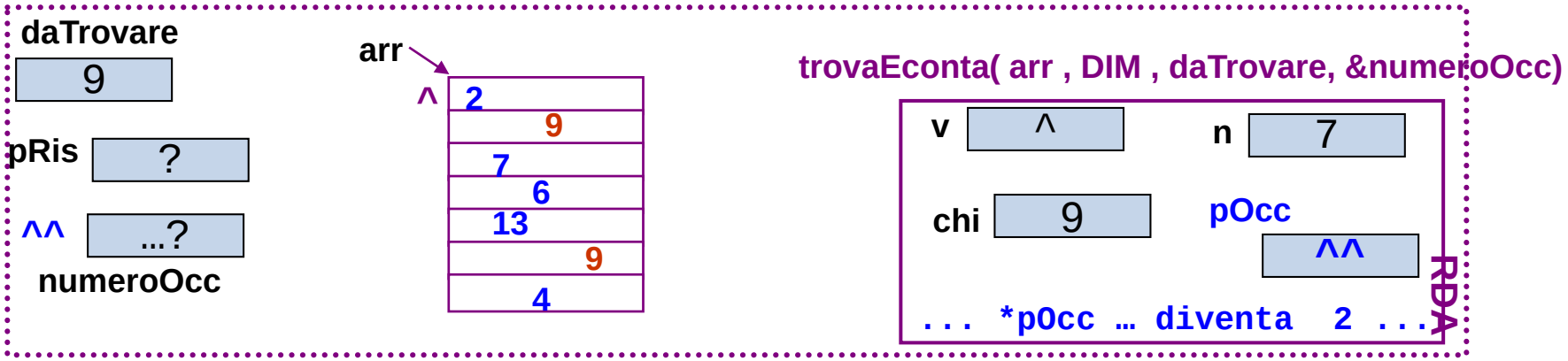


Uso di puntatori per PARAMETRI DI OUTPUT - 3/9 -

```
pRis = trovaEconta( arr , DIM , daTrovare , &numeroOcc)
```

- la funzione riceve l'indirizzo della variabile esterna numeroOcc (di proprieta` della funzione chiamante) e usandolo riesce a modificare quella variabile!
- la definizione della funzione, con un parametro formale "di output" sara`

```
int * trovaEconta (int * v, int n, int chi, int *pOcc) {
…
}
```

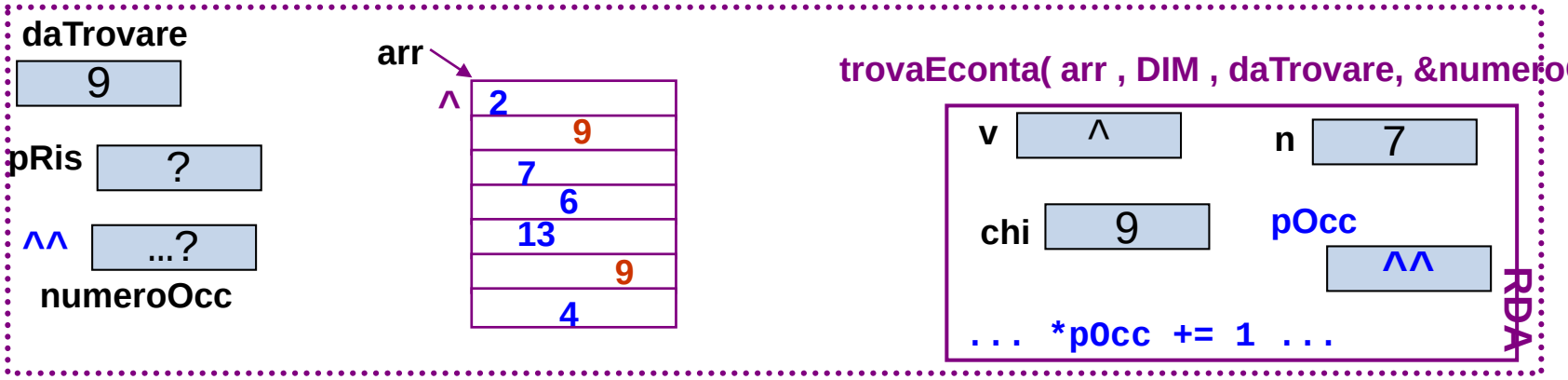


Uso di puntatori per PARAMETRI DI OUTPUT - 4/9 -

```
pRis = trovaEconta( arr , DIM , daTrovare , &numeroOcc)
```

- la funzione riceve l'indirizzo della variabile esterna numeroOcc (di proprieta` della funzione chiamante) e usandolo riesce a modificare quella variabile!
- la definizione della funzione, con un parametro formale "di output" sara`

```
int * trovaEconta (int * v, int n, int chi, int *pOcc) {
…
}
```



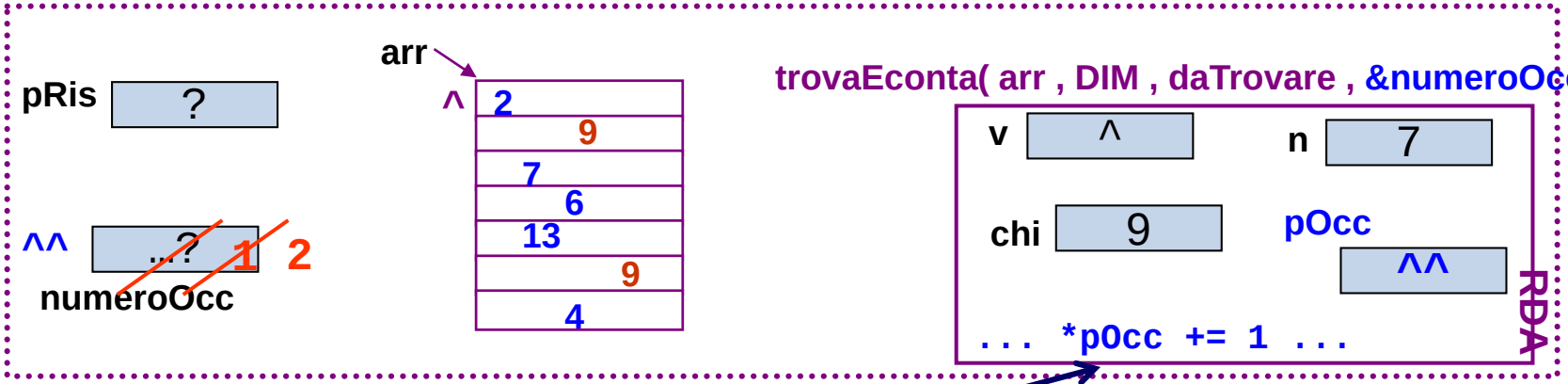
Uso di puntatori per PARAMETRI DI OUTPUT – 5/9 –

La soluzione, per gestire PARAMETRI DI OUTPUT, e` nell'uso della tecnica del PASSAGGIO DI INDIRIZZO:

...

- la funzione riceve l'indirizzo della variabile esterna numeroOcc (di proprieta` della funzione chiamante) e usandolo riesce a modificare quella variabile!
- la definizione della funzione, con un parametro formale "di output" sara`

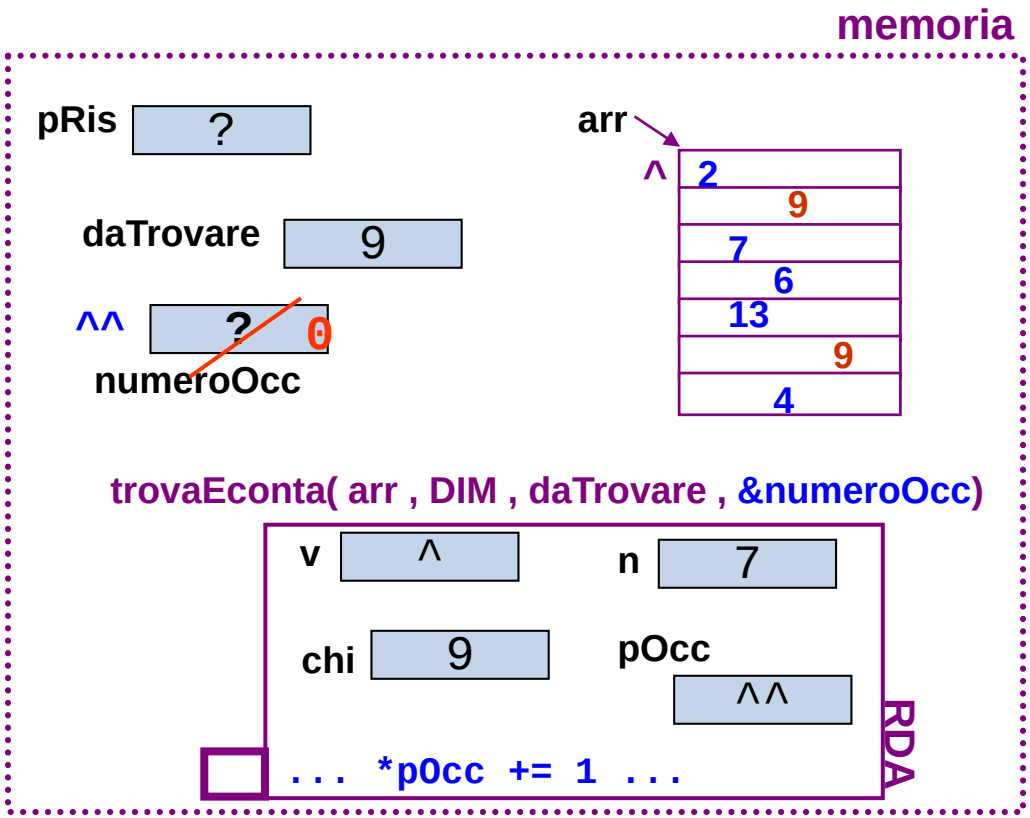
```
int * trovaEconta (int * v, int n, int chi, int *pOcc) {
…
}
```



con questa istruzione, eseguita nell'ambiente della funzione, viene modificata una locazione che appartiene ad un ambiente diverso

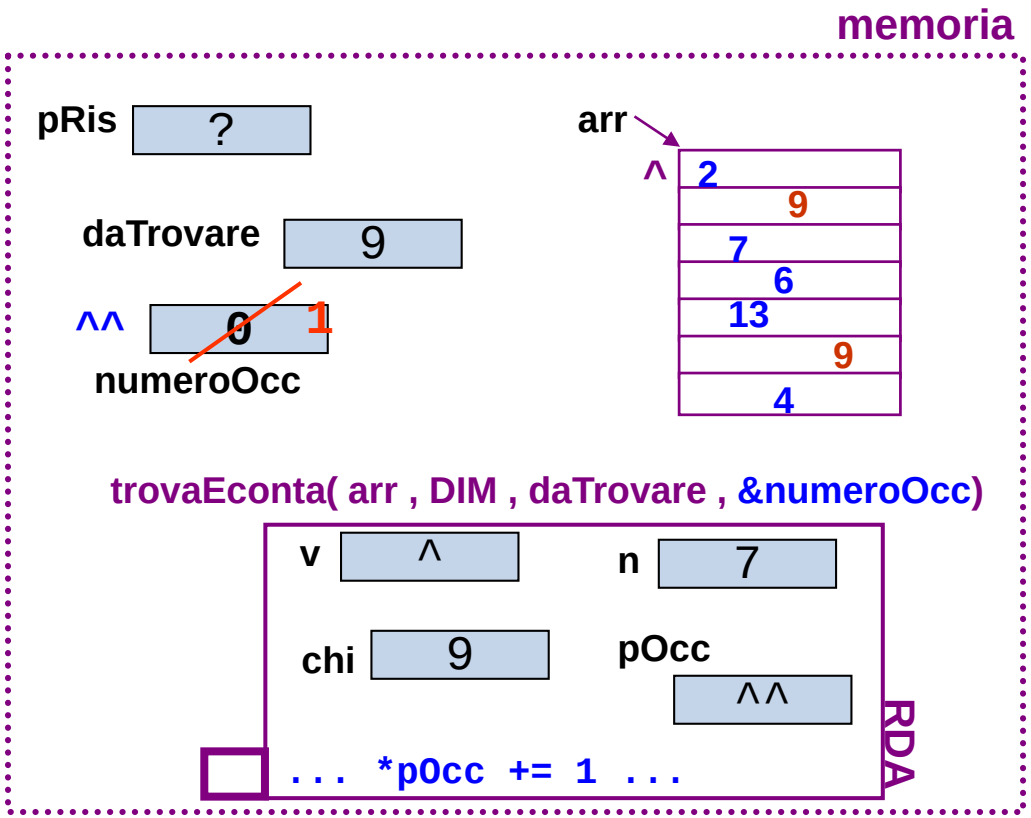
Uso di puntatori per PARAMETRI DI OUTPUT - 6/9 -

```
/* definizione funzione */
int * trovaEconta (int *v, int n, int chi, int *pOcc)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL    */
        *pi = v;       /* per scandire array       */

 *pOcc = 0;            /* contatore occorrenze NB da inizializzare!*/

 while( pi < v+n) {

     if (*pi == chi) {
       pR = pi;
       *pOcc += 1;
     }

     pi++;
 }

return pR;
}
```



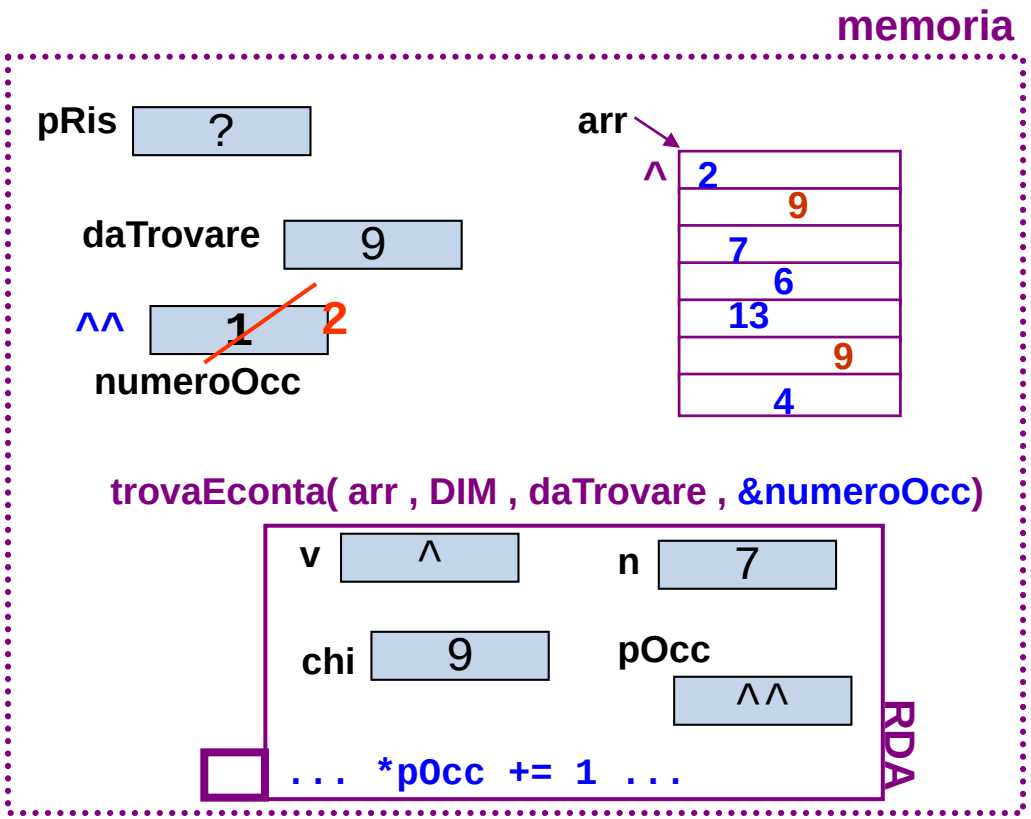
Uso di puntatori per PARAMETRI DI OUTPUT - 7/9 -

```
/* definizione funzione */
int * trovaEconta (int *v, int n, int chi, int *pOcc)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL    */
        *pi = v;       /* per scandire array       */

 *pOcc = 0;            /* contatore occorrenze NB da inizializzare!*/

 while( pi < v+n) {

     if (*pi == chi) {
       pR = pi;
       *pOcc += 1;
     }

     pi++;
 }

return pR;
}
```



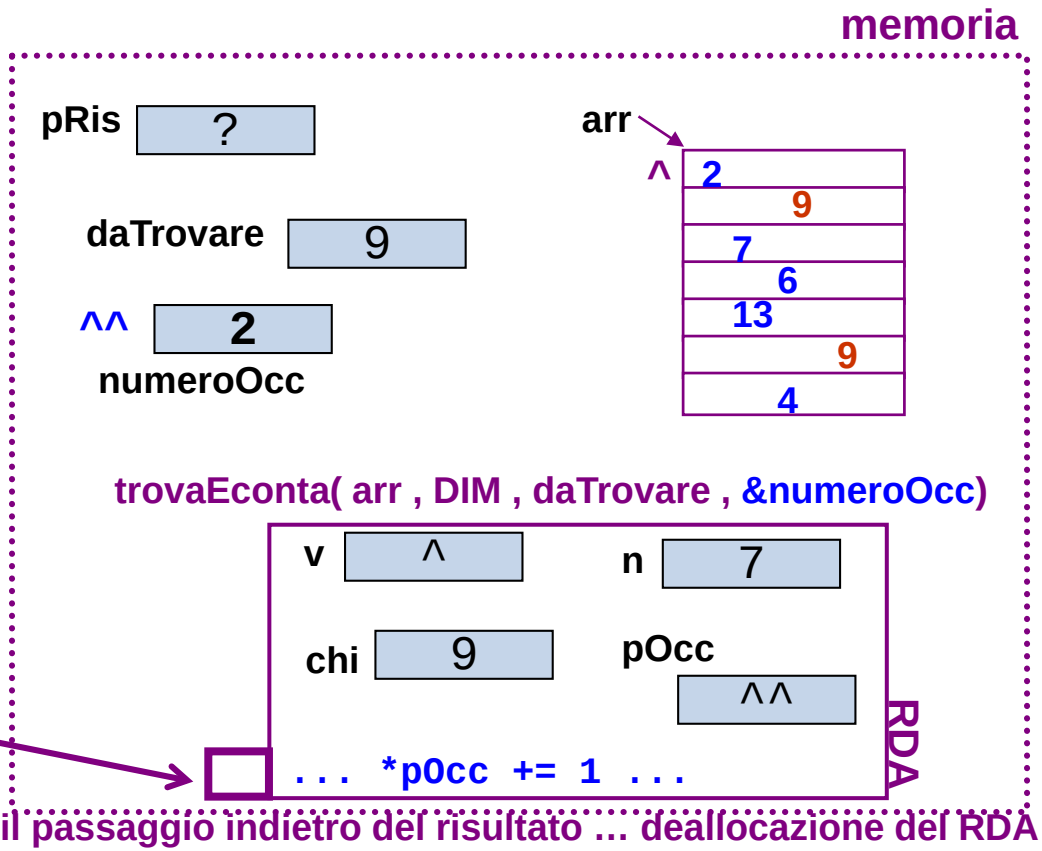
Uso di puntatori per PARAMETRI DI OUTPUT - 8/9 -

```
/* definizione funzione */
int * trovaEconta (int *v, int n, int chi, int *pOcc)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL    */
        *pi = v;       /* per scandire array       */

 *pOcc = 0;            /* contatore occorrenze NB da inizializzare!*/

 while( pi < v+n) {

     if (*pi == chi) {
       pR = pi;
       *pOcc += 1;
     }

     pi++;
 }

return pR;
}
```



Uso di puntatori per PARAMETRI DI OUTPUT - 9/9 -

```
/* definizione funzione */
int * trovaEconta (int *v, int n, int chi, int *pOcc)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL    */
        *pi = v;       /* per scandire array       */

 *pOcc = 0;            /* contatore occorrenze NB da inizializzare!*/

 while( pi < v+n) {

     if (*pi == chi) {
       pR = pi;
       *pOcc += 1;
     }

     pi++;
 }

return pR;
}
```



Uso di puntatori per PARAMETRI DI OUTPUT - 10/9 -

```
/* definizione funzione */
int * trovaEconta (int *v, int n, int chi, int *pOcc)
    int *pR = NULL,    /* risultato (se non cambia
                          mai restituiremo NULL    */
        *pi = v;       /* per scandire array       */

 *pOcc = 0;            /* contatore occorrenze NB da inizializzare!*/

 while( pi < v+n) {

     if (*pi == chi) {
```

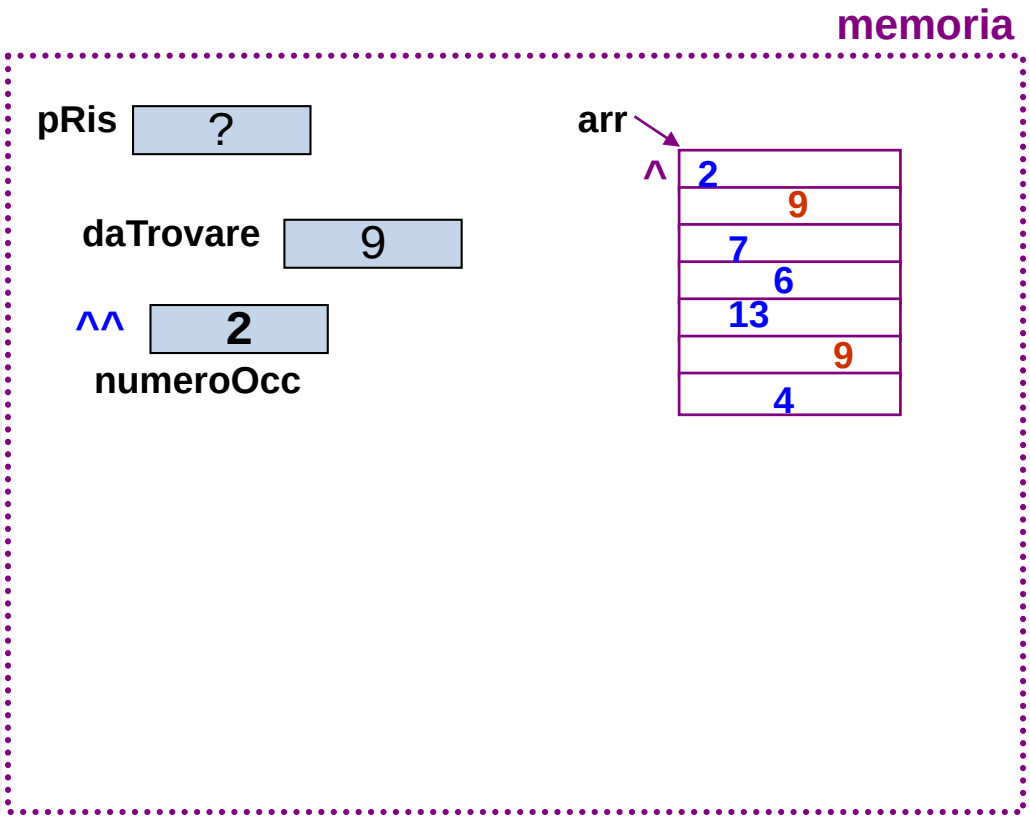
situazione finale …

la chiamata di trovaEconta, nell'istruzione

```
pRis = trovaEconta(arr, DIM,
daTrovare, &numeroOcc);
```

e` stata eseguita, il risultato si assegna a pRis, mentre numeroOcc ha subito "effetti collaterali" dalla esecuzione della chiamata ed e` diventato 2.

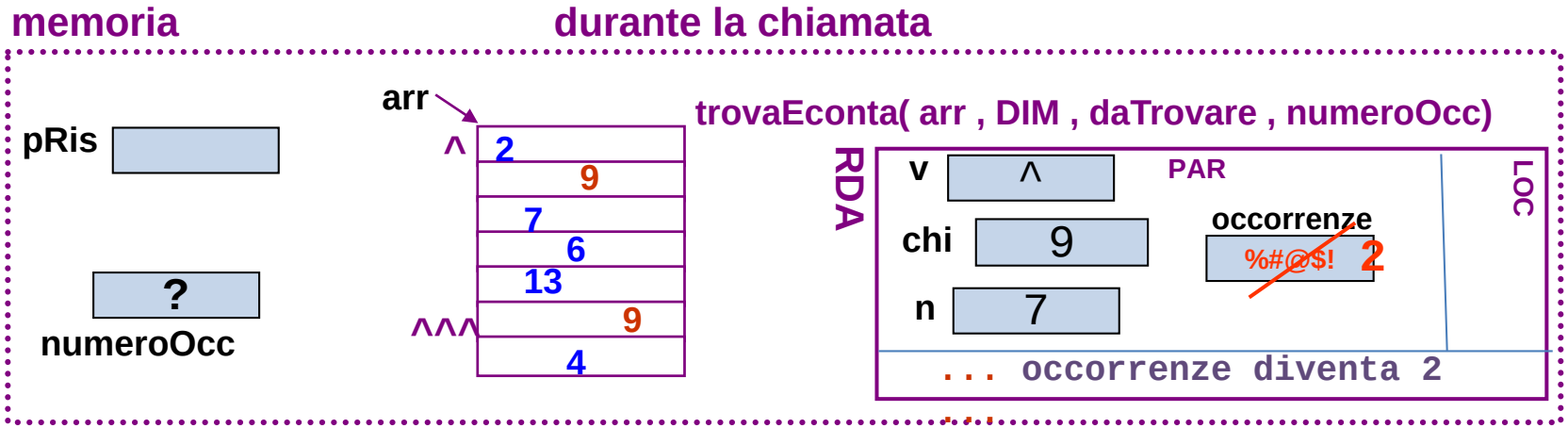
E poi si prosegue con il resto della funzione chiamante.



Riflessione: ma perche' usare i puntatori? Non si puo' passare numeroOcc e basta? (1/3)

```
int * trovaEconta (int * v, int n, int chi, int occorrenze) {
…
}
```

```
int main() {   int     daTrovare = 9, arr[DIM],
                                numeroOcc,              *pRis;
...
  pRis = trovaEconta(arr, DIM, daTrovare, numeroOcc);
...
```



Riflessione: ma perche' usare i puntatori? Non si puo' passare numeroOcc e basta? (2/3)

No … cosi` non va … in qualche modo numeroOcc deve essere un parametro della funzione, o collegato ad un parametro, in modo che una istruzione della funzione lo possa modificare. Ma come?

```
int * trovaEconta (int * v, int n, int chi, int occorrenze) {
…
}
```

(la risposta e` no ...)

```
int main() {   int     daTrovare = 9
                                numero
...
  pRis = trovaEconta(arr, DIM, daTrovare, numeroOcc);
...
```

abbiamo modificato un parametro formale, dentro al RDA … poi l'RDA sparisce e il parametro formale occorrenze con lui … non abbiamo modificato la variabile numeroOcc della funzione chiamante …

memoria — dopo la chiamata

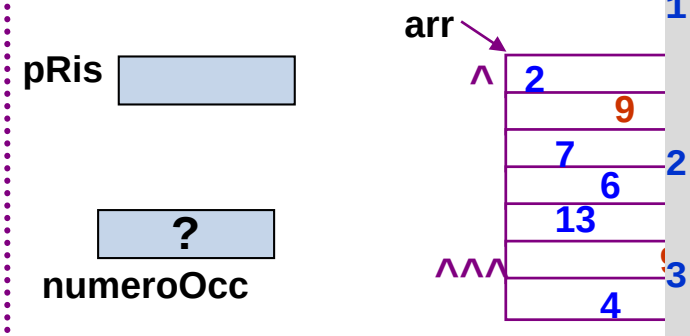
Riflessione: ma perche' usare i puntatori? Non si puo' passare numeroOcc e basta? (3/3)

No … cosi` non va … in qualche modo numeroOcc deve essere un parametro della funzione, o collegato ad un parametro, in modo che una istruzione della funzione lo possa modificare. Ma come?

```
int * trovaEconta (int * v, int n, int chi, int occorrenze) {
…
}

int main() {   int     daTrovare = 9, arr[DIM],
                              numeroOcc,              *pRis;
...
  pRis = trovaEconta(arr, DIM, daTrovare, numeroOcc);
...
```

abbiamo modificato un parametro formale, dentro al RDA … poi l'RDA sparisce e il parametro formale occorrenze con lui … non abbiamo modificato la variabile numeroOcc della funzione chiamante …

memoria — **dopo la chiamata**



1) il parametro occorrenze corrisponde ad una locazione interna al RDA (DIVERSA DALLA numeroOcc della main()
2) il passaggio di un parametro e` "per valore" (QUI valore non definito!!)
3) occorrenze (parametro formale) viene modificata. La numeroOcc della main() NO.