Lezione 4

Ancora sulle funzioni: stack di attivazione di funzione
Array: indirizzamento. Parametri array(?)
Side effects
Eliminazione elemento array

Informatica, Corso B - D. Tamascelli

Riassunto puntata precedente

- Tipo delle operazioni + cast
- Funzione/procedura: dichiarazione, definizione, parametri formali, parametri attuali
- Array: che cosa sono, dichiarazione, uso (accesso agli elementi)

Oggi

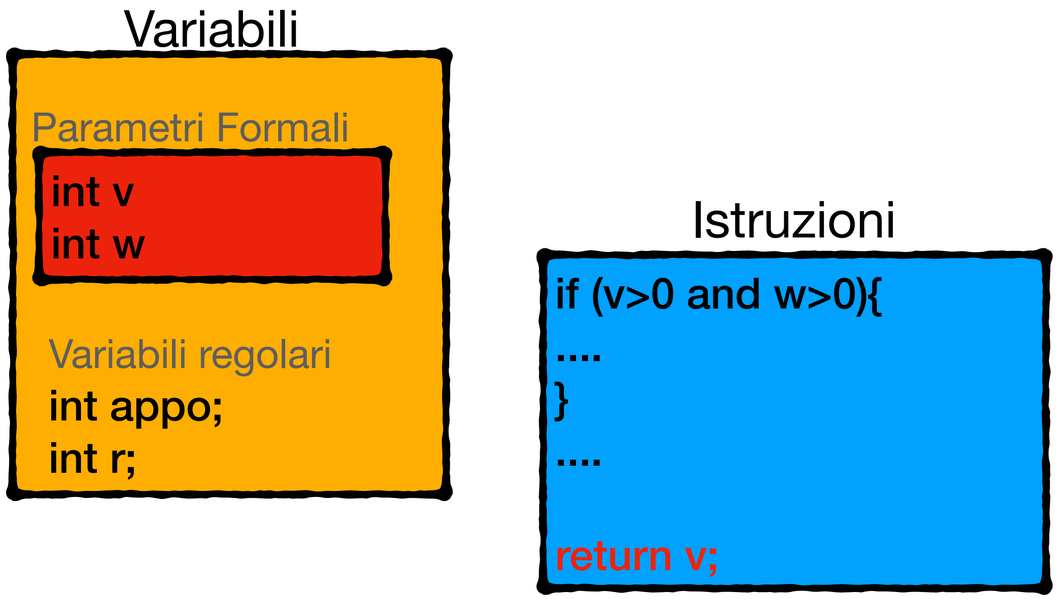
- Ancora sulle funzioni: record e stack di attivazione di funzione/procedura
- "Huston....we have a problem..."
- Array: che cosa sono...davvero....
- Applicazione

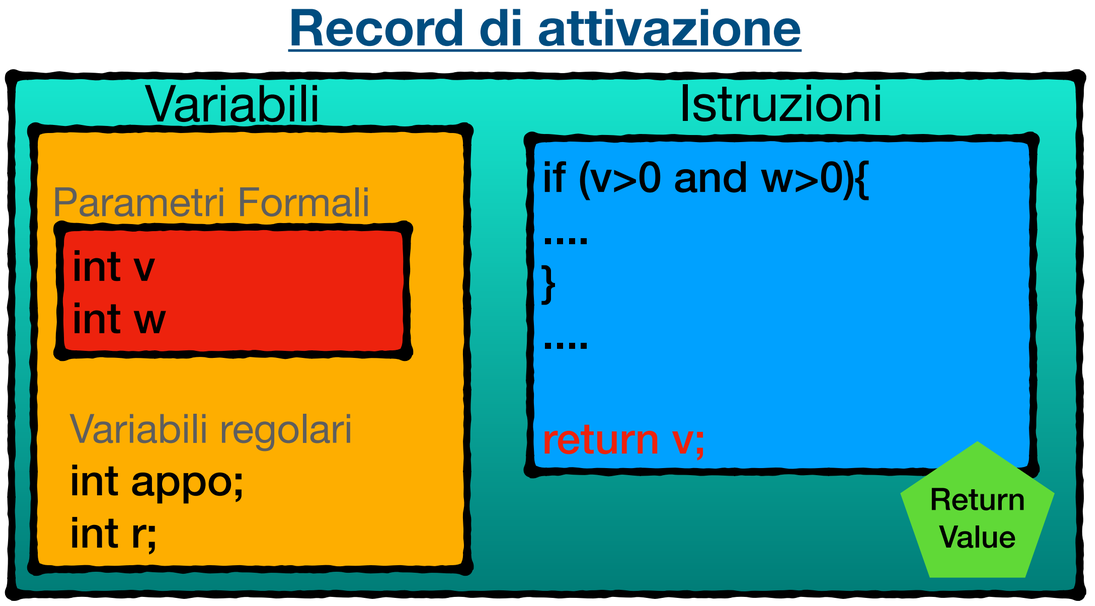
Record di attivazione

```
int MCD(int v, int w){

    int appo;
    int r;

    if (v>0 and w>0){
```

....

```
return v;
```





Record di attivazione

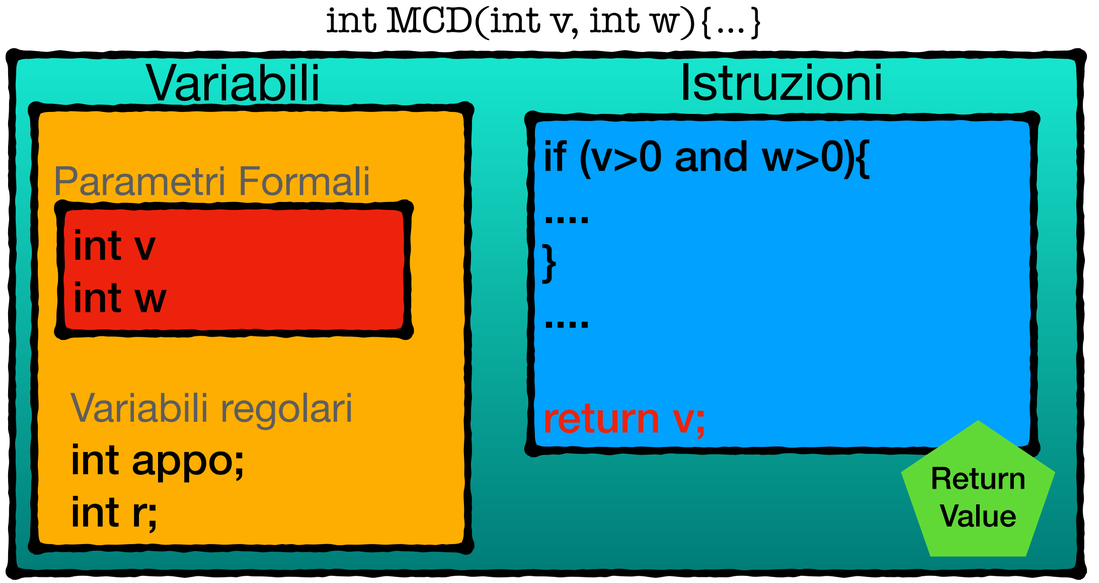
La definizione di ogni funzione/procedura porta alla creazione di una particolare struttura,

Record di Attivazione (di funzione/procedura)

che contiene:

- Tutte le Variabili (inclusi i parametri formali) della funzione.
- Tutte le Istruzioni della funzione.
- Una locazione particolare per il Return Value.

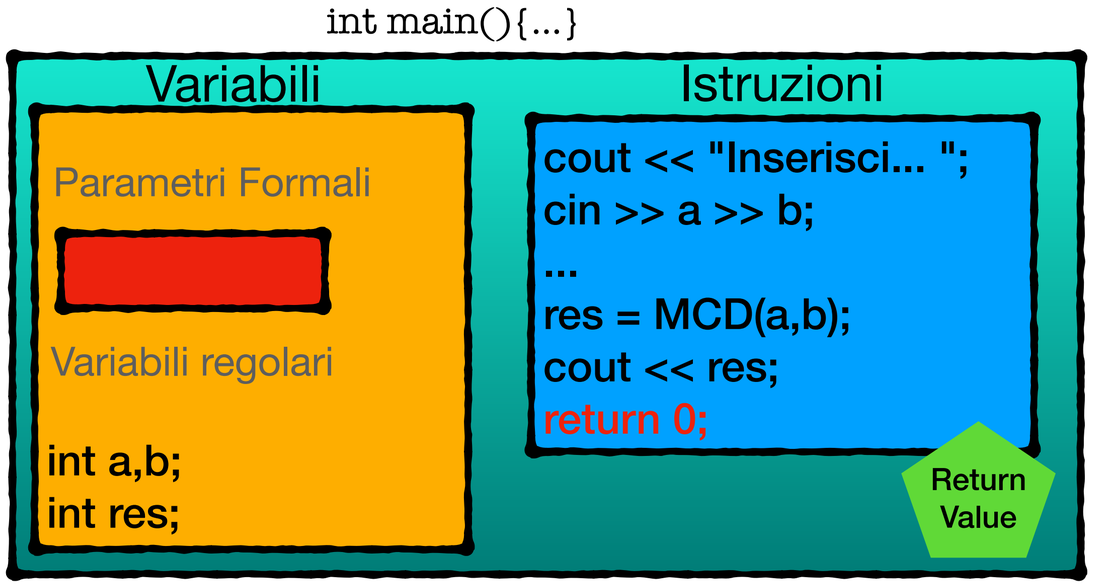
Record di attivazione



Osservazioni

- Le Variabili sono TUTTE Locali: vivono solo nel record di attivazione.
- Tutte le Istruzioni della funzione hanno accesso diretto solo alle variabili (informazioni) locali della funzione.
- L'istruzione di `return` è sempre l'ultima eseguita dalla funzione: comporta la scrittura del Return Value

Record di attivazione main



Nota

- Anche main è una funzione
- In quanto tale comporta la definizione di un record di attivazione.

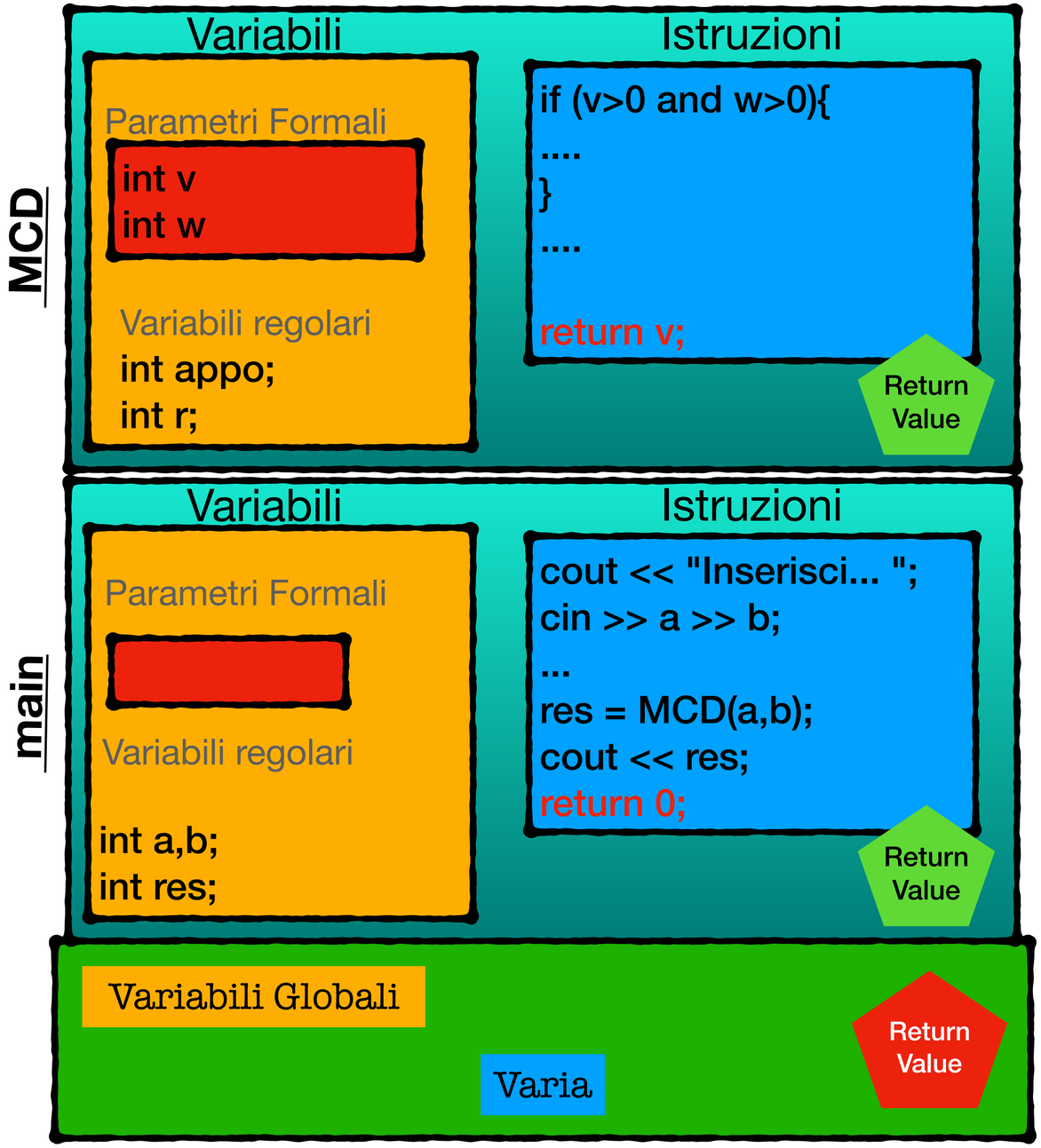
Stack di attivazione (1)

Come funziona l'esecuzione di un programma?

- Programma chiamato:

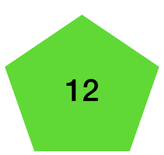
```
(base) tama@TamaAir ~ % ./calcolaMCD.x
```

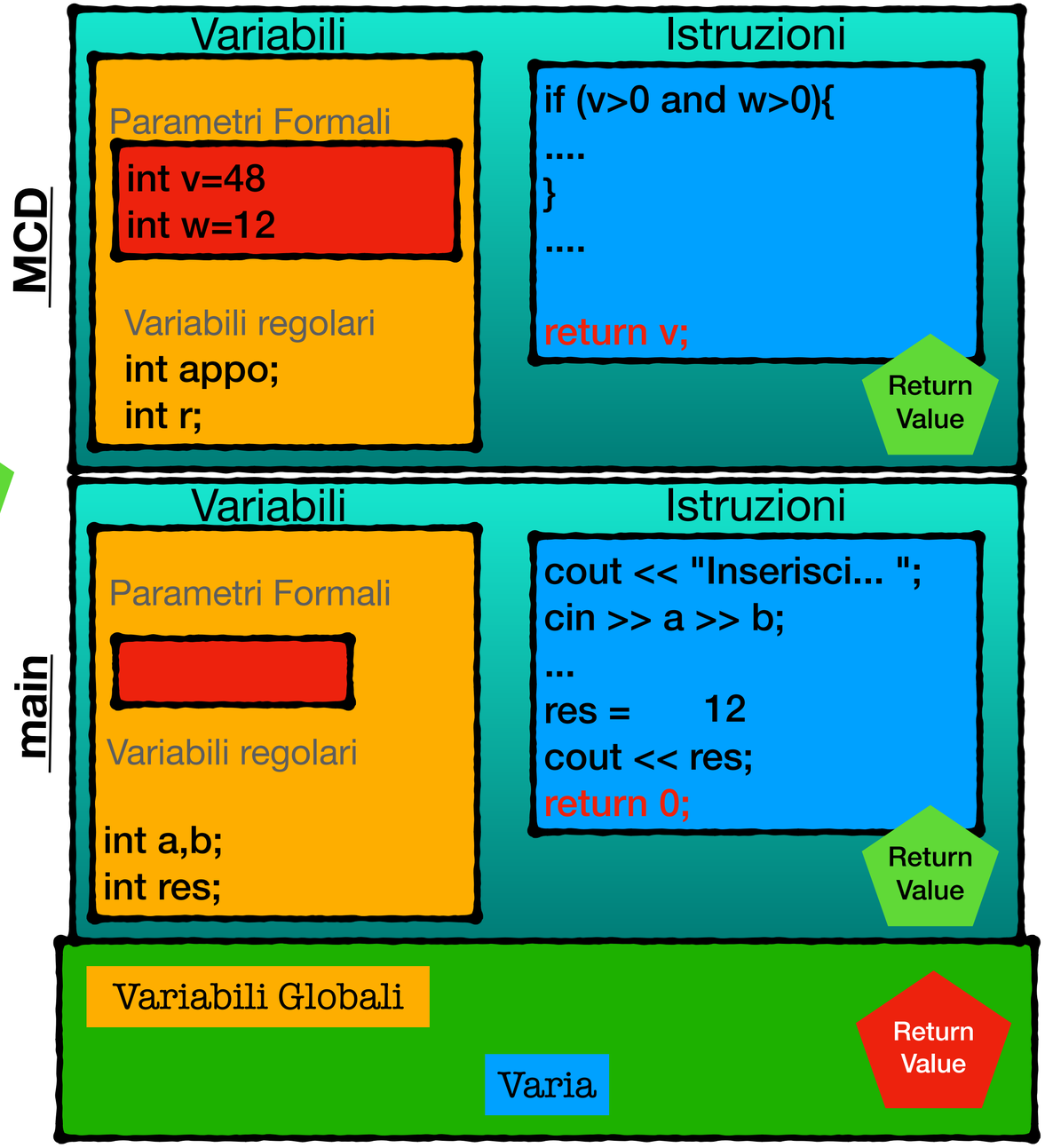
- L'esecutore (SO):
  - assegna al programma una zona di memoria, dove colloca un record contenente informazioni che servono...
  - "Carica" il record di attivazione del main....
  - Comincia ad eseguire le istruzioni del main
  - Il RHS di un assegnamento è una funzione: l'esecuzione del flusso del main deve essere interrotta (serve un valore!)
  - Viene caricato il record di attivazione di MCD
  - I parametri formali vengono inizializzati usando i valori dei parametri attuali.



Stack di attivazione (2)

- Supponiamo a=48, b =12....
- Inizializzate le variabili locali/parametri formali usando i valori degli omologhi parametri attuali...
- Comincia l'esecuzione delle istruzioni di MCD
- Quando si incontra l'istruzione return v (v=12)
- Il valore viene scritto nel return value
- Il return value viene scritto al posto della chiamata di MCD
- Il record di attivazione di MCD viene cancellato/ rimosso/eliminato
- Riprende l'esecuzione delle istruzioni del main.
- Quando si incontra il return nel main il valore (0) viene scritto nel return value....
- ...e il programma termina, restituendo al chiamante il valore 0 (o quello che sarà....)





Osservazioni (1)

La chiamata di una funzione/procedura, provoca:
- l'interruzione dell'esecuzione delle istruzioni della funzione/procedura chiamante
- l'allocazione del record di attivazione della funzione/procedura chiamata

Ogni funzione ha accesso solo alle informazioni contenute nelle sue variabili locali

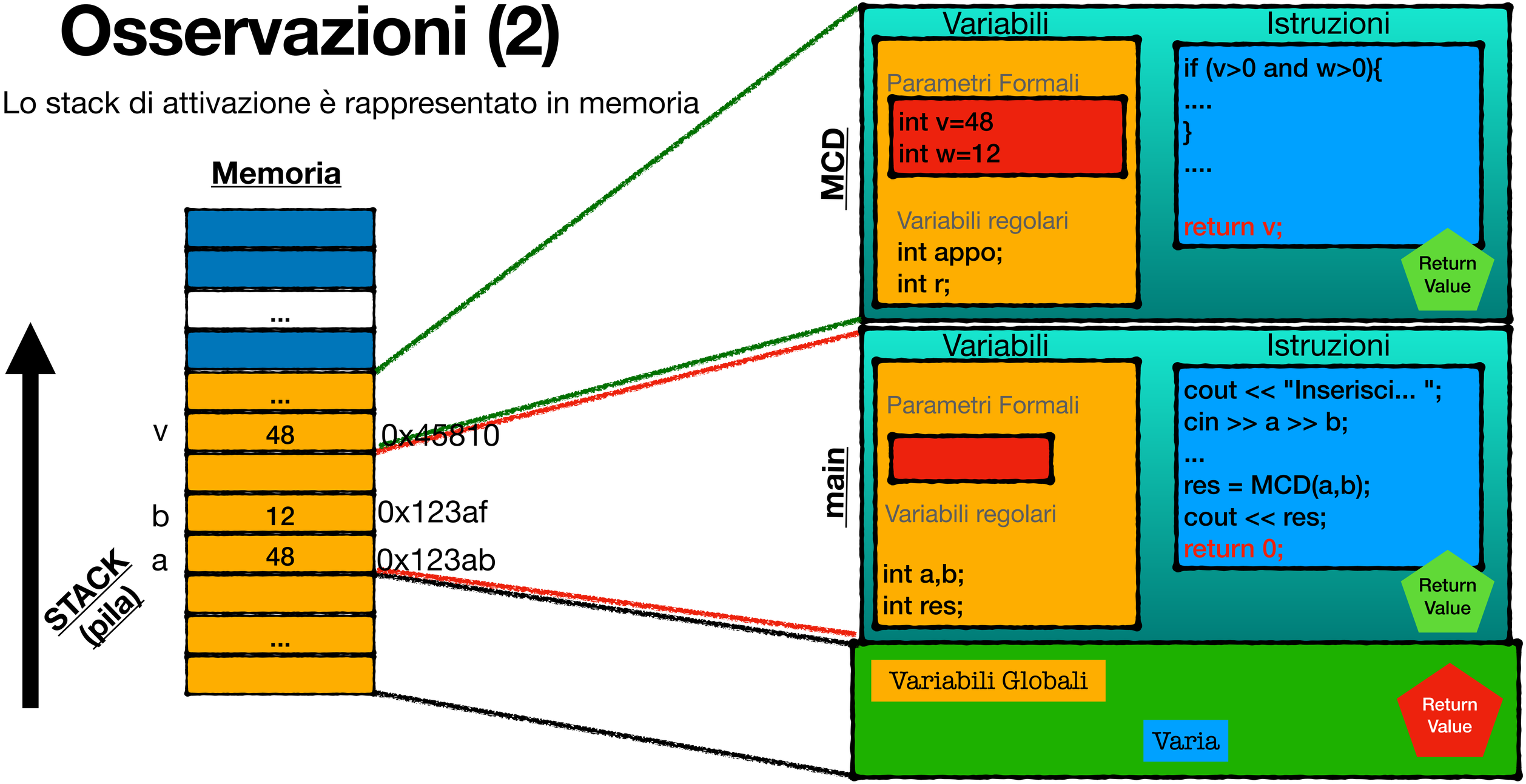
Lo scambio di informazioni tra funzioni è limitato a:
- passaggio di parametri (inizializzazione parametri formali)
- restituzione di valori (di ritorno)

Una funzione restituisce al più un solo valore, di un certo tipo.
Se non restituisce alcun valore, si chiama **procedura**.

L'esecuzione di un programma parte e finisce contestualmente alla presenza del record di attivazione del main

Osservazioni (2)

Lo stack di attivazione è rappresentato in memoria



Esercizio

Simulazione stack di attivazione del programma che calcola:
- Media
- Deviazione

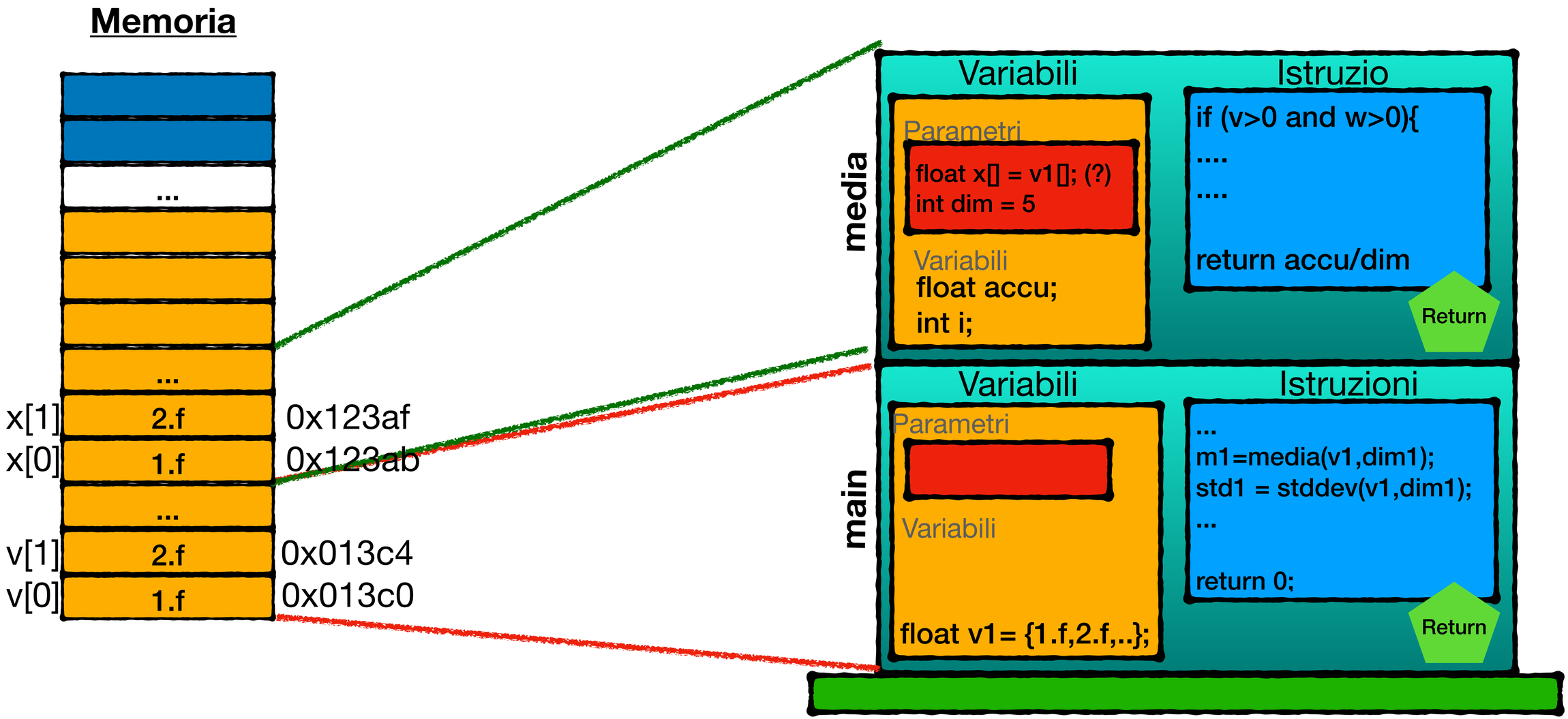
dei dati contenuti in un array

```c
int main(){
    float v1[5] = {1.f,2.f,3.f,4.f,5.f};
    float v2[4]={11.f,12.f,13.f,14.f};
    int dim1 = 5;
    int dim2 = 4;
    float m1,m2;
    float std1,std2;

    m1 = media(v1,dim1);
    m2 = media(v2,dim2);
    std1 = stddev(v1,dim1);
    std2 = stddev(v2,dim2);
```

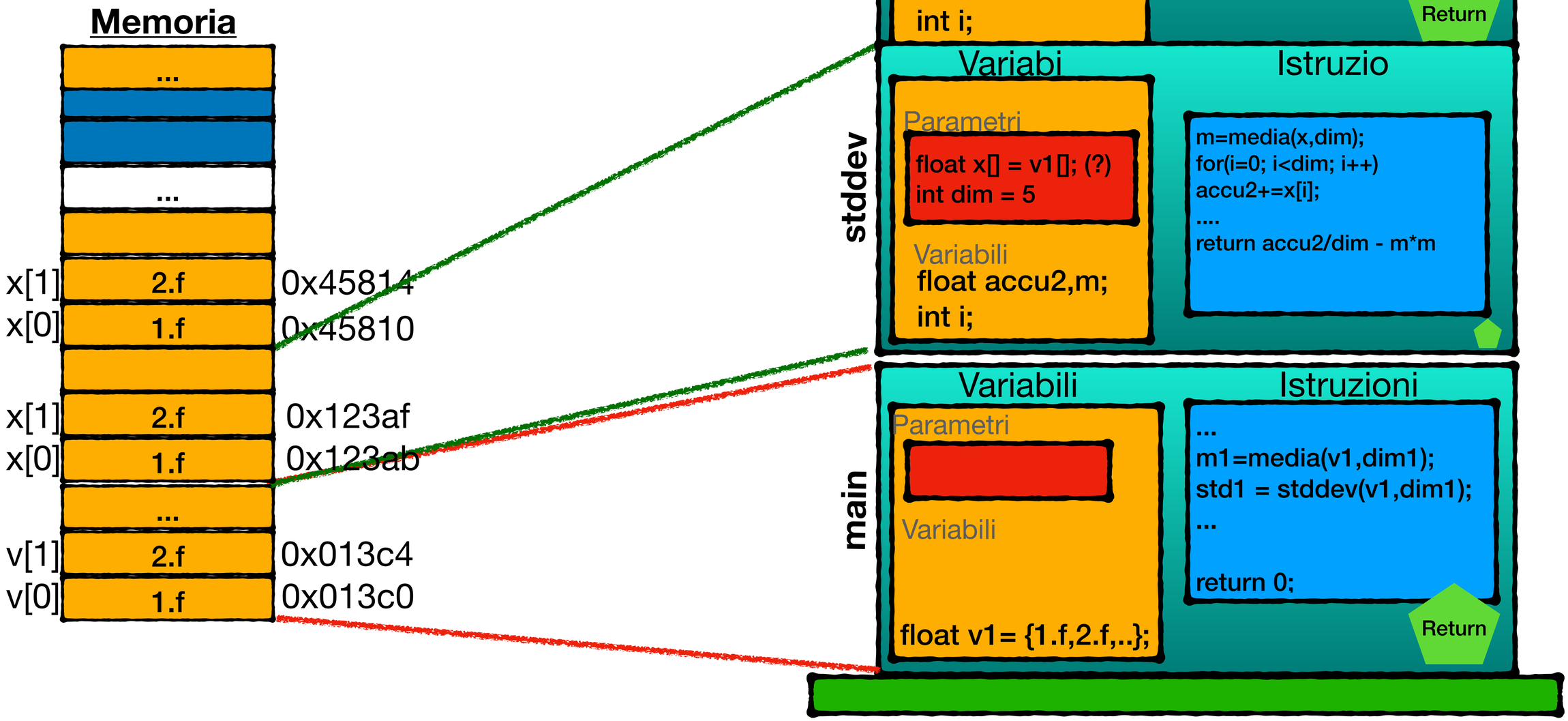
```c
float media(float x[], int n){

    float accu = 0.f;
    int i = 0;

    while(i < n){
        accu  = accu + x[i];
        i++; //aka i=i+1;
    }

    //Valore restituito dalla funzione.
    return accu/n;
}
```

```c
float stddev(float x[], int n){

    float accu = 0;

    float m;

    int i = 0;

    m = media(x,n);

    while(i<n){
        accu  = accu +pow(x[i] - m,2);
        i++;
    }

    return sqrt(accu/n);
}
```

Esercizio (2)

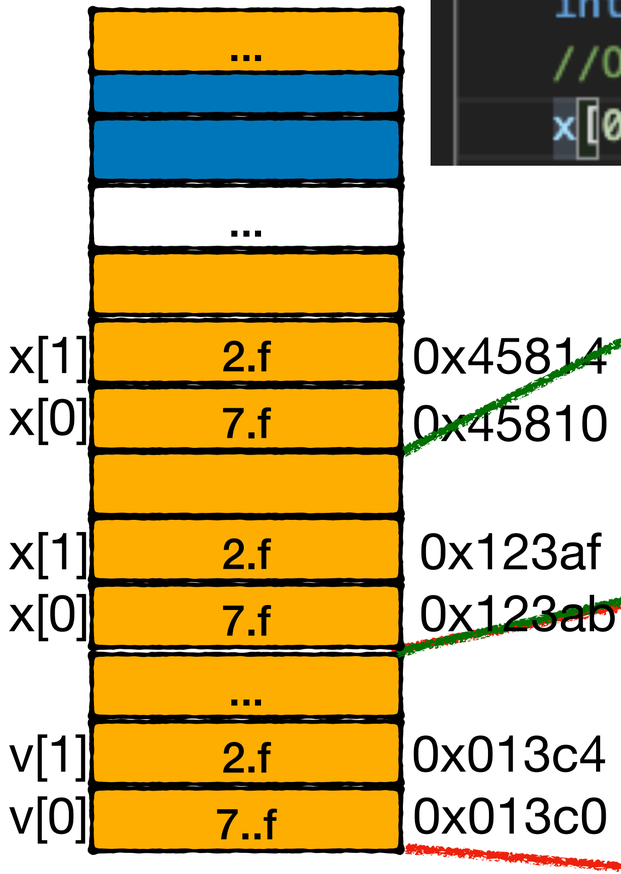

Memoria
...
...
x[1]
2.f
0x123af
x[0]
1.f
0x123ab
...
v[1]
2.f
0x013c4
v[0]
1.f
0x013c0
media
Variabili
Istruzio
Parametri
float x[] = v1[]; (?)
int dim = 5
Variabili
float accu;
int i;
if (v>0 and w>0){
....
....
return accu/dim
Return
main
Variabili
Istruzioni
Parametri
Variabili
float v1= {1.f,2.f,..};
...
m1=media(v1,dim1);
std1 = stddev(v1,dim1);
...
return 0;
Return

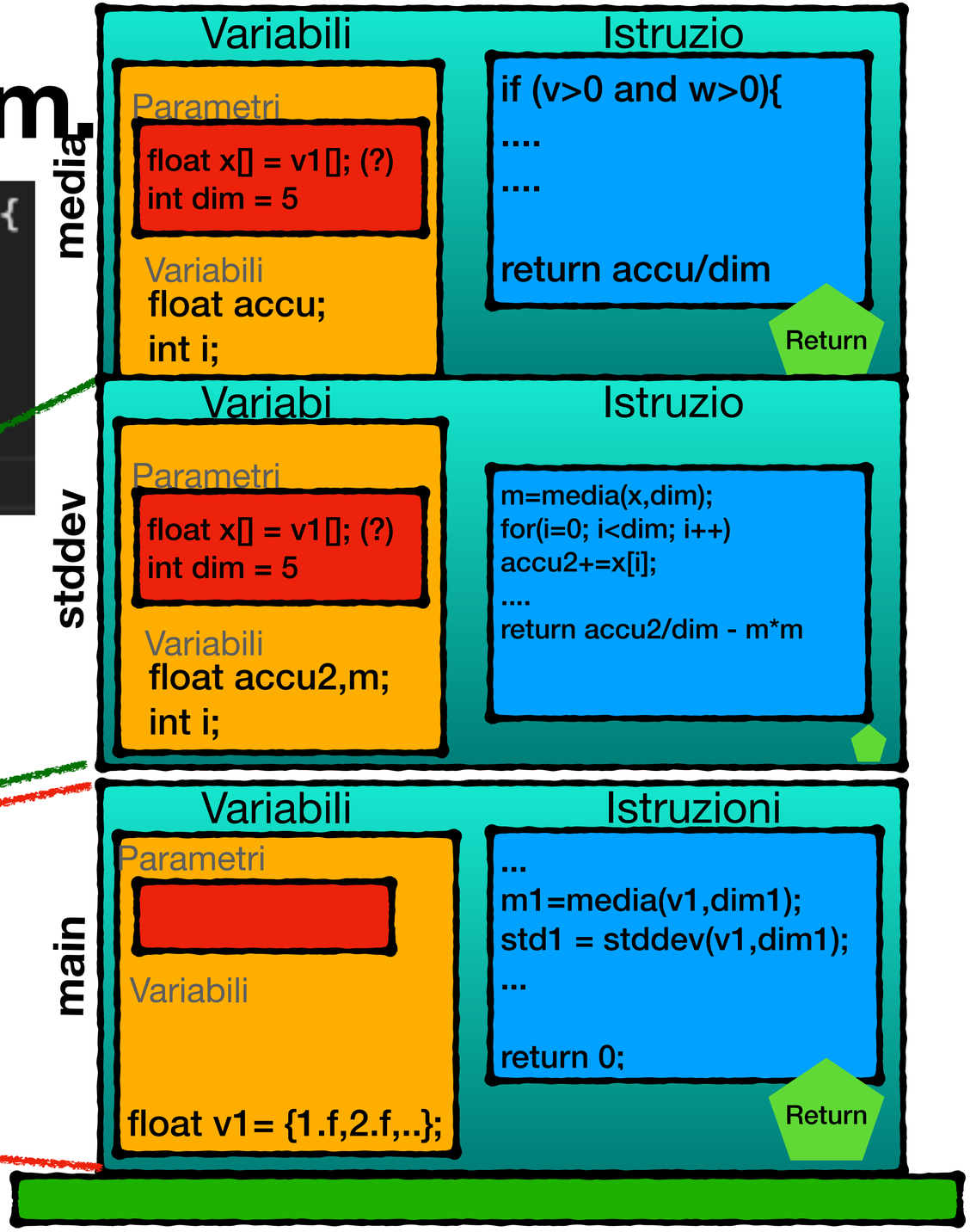
Esercizio (3)


Memoria
...
...
x[1] 2.f 0x45814
x[0] 1.f 0x45810
x[1] 2.f 0x123af
x[0] 1.f 0x123ab
...
v[1] 2.f 0x013c4
v[0] 1.f 0x013c0
media
Variabili
Istruzio
Parametri
float x[] = v1[]; (?)
int dim = 5
Variabili
float accu;
int i;
if (v>0 and w>0){
....
....
return accu/dim
Return
stddev
Variabi
Istruzio
Parametri
float x[] = v1[]; (?)
int dim = 5
Variabili
float accu2,m;
int i;
m=media(x,dim);
for(i=0; i<dim; i++)
accu2+=x[i];
....
return accu2/dim - m*m
main
Variabili
Istruzioni
Parametri
Variabili
float v1= {1.f,2.f,..};
...
m1=media(v1,dim1);
std1 = stddev(v1,dim1);
...
return 0;
Return

Huston...we have a problem

```
float media(float x[], int n){

    float accu = 0.f;
    int i = 0;
    //Occhio
    x[0] = 7.f;
```





Quindi...?

Ogni funzione ha accesso solo alle informazioni contenute nelle sue variabili locali

Questo principio sembra essere violato: la modifica fatta all'interno di una funzione si propaga al di fuori della funzione.

Il problema è che vi ho raccontato una frottola: i parametri formali array non sono mai stati inizializzati copiando, membro a membro, gli omologhi array attuali!

Al di là di ogni considerazione sulla morale del docente: che cosa succede davvero?

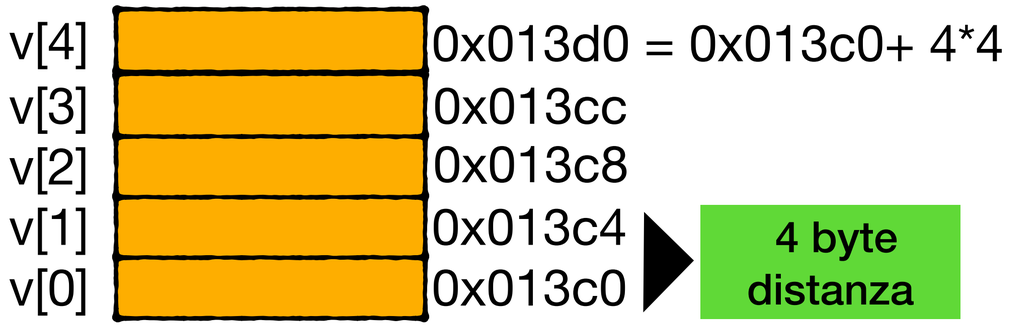
I parametri array "contengono" l'indirizzo in cui comincia l'array (in questo caso di float).

In pratica le funzioni accedono al contenuto dell'array originale....da sempre....

Array: che cosa sono...davvero

- n-upla ordinata di elementi dello stesso tipo, supponiamo `float`
- la dichiarazione contiene tutte le informazioni necessarie a preparare l'array

```
float v[5];
```



- Servono 5 * sizeof(float) = 5 *4 =20 byte per rappresentare 5 float.
- Se i float vengono disposti in celle di memoria contigue...
- ...posso recuperare l'i-esimo elemento sapendo dove inizia l'array, e sommando i*sizeof(float)
- se indichiamo con &v[0] la base (punto di partenza dell'array), ovvero l'indirizzo del primo elemento dell'array:

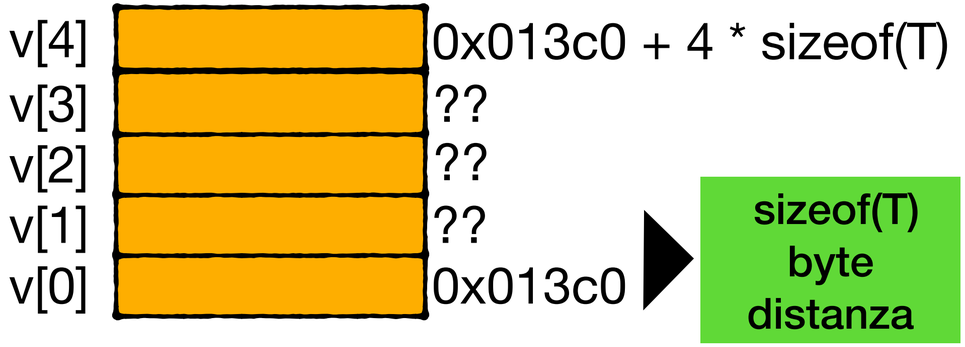
$$v[i] \longleftarrow base + i * sizeof(float)$$

- Se ad una funzione passassimo l'informazione "**indirizzo** a cui un array comincia"...
- ...e anche il **tipo** degli elementi dell'array,
- la funzione saprebbe come accedere agli elementi dell'array.,

Array: che cosa sono...davvero

- n-upla ordinata di elementi dello stesso tipo, che indichiamo genericamente con `T`
- la dichiarazione contiene tutte le informazioni necessarie a preparare l'array

```
T v[5];
```



- Servono 5 * sizeof(`T`) byte per rappresentare 5 `dati di tipo T`.
- Se i dati di tipo T vengono disposti in celle di memoria contigue...
- ...posso recuperare l'i-esimo elemento sapendo dove inizia l'array, e sommando i*sizeof(T)
- se indichiamo con &v[0] la base (punto di partenza dell'array), ovvero l'indirizzo del primo elemento dell'array:

$$v[i] \longleftarrow base + i * sizeof(T)$$

- Se ad una funzione passassimo l'informazione "**indirizzo** a cui un array comincia"...
- ...e anche il **tipo** degli elementi dell'array,
- la funzione saprebbe come accedere agli elementi dell'array.,

Quindi...?

Ogni funzione ha accesso solo alle informazioni contenute nelle sue variabili locali

Questo principio sembra essere ora ripristinato: alla funzione arriva in effetti un'informazione particolare: l'indirizzo in cui comincia l'array

Vi ho raccontato una frottola...ma ho fatto ammenda. Lasciamo però per ora sospeso il problema di come un "indirizzo di memoria" possa essere passato...

Dal punto di vista pratico:

I parametri array "contengono" l'indirizzo in cui comincia l'array (in questo caso di float).

In pratica le funzioni accedono al contenuto dell'array originale....da sempre....

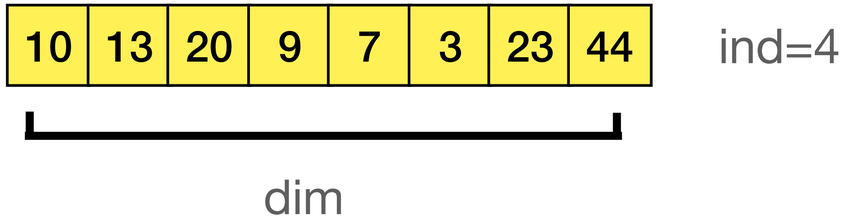
Se stiamo attenti a non modificare i valori nell'array, possiamo anche ignorare la cosa...

Alcuni semplici algoritmi...su array.

Eliminazione di un elemento da un array

Input (Istanza)

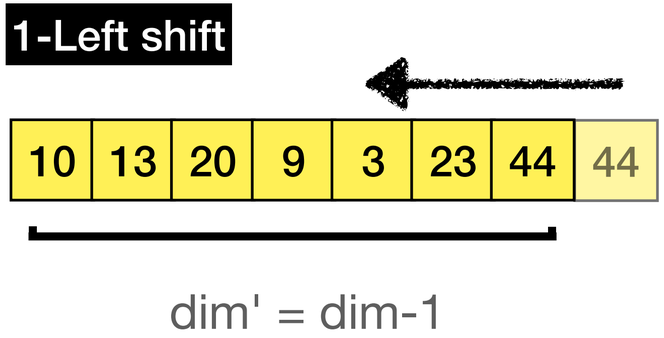
- array (di tipo T)
- dimensione array
- posizione elemento da eliminare

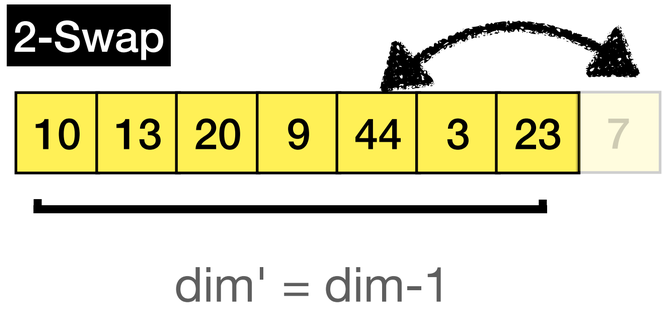


Output

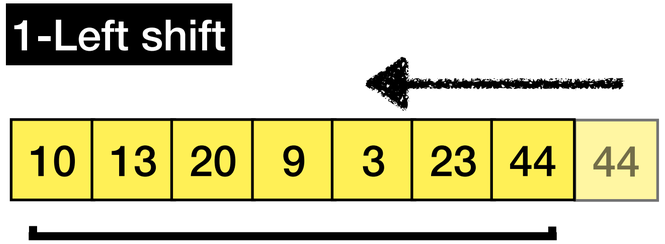
- array (di tipo T)
- nuova dimensione dell'array (-1 se problema)

Possibili soluzioni





Soluzioni: dettaglio

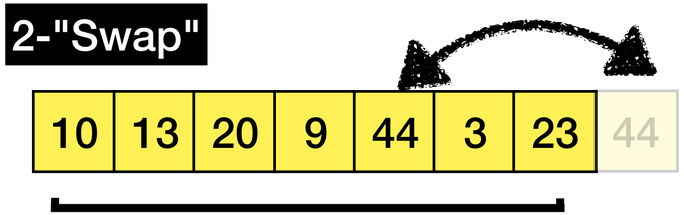


dim' = dim-1

INPUT: (x[],dim,ind)
se (ind >dim-1 o ind <0) //errore
    OUTPUT: -1

altrimenti:
    //Left shift
    per i = ind,...,dim-2:
        x[i] = x[i+1];

    OUTPUT: dim-1



dim' = dim-1

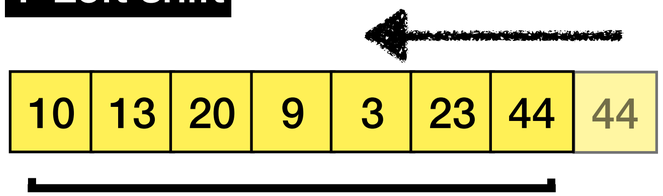
INPUT: (x[],dim,ind)
se (ind >dim-1 o ind <0) //errore
    OUTPUT: -1

altrimenti:
//Sovrascrivi ind con l'ultimo valore
//dell'array.
per i = ind,...,dim-2:
    x[ind] = x[dim-1];

OUTPUT: dim-1

Confronto soluzioni

1-Left shift

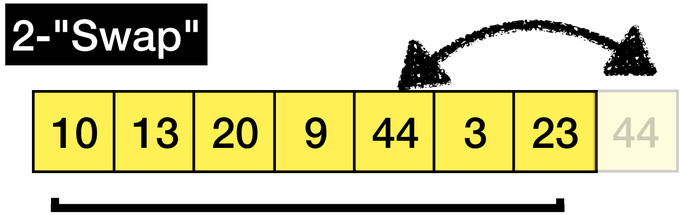


dim' = dim-1

Caratteristiche:

- Richiede, in media, dim/2 scritture per effettuare lo shift.
- Preserva l'ordine relativo degli elementi dell'array.

2-"Swap"



dim' = dim-1

Caratteristiche:

- Richiede, una sola scrittura.
- Non preserva l'ordine relativo degli elementi.

Osservazioni

Una funzione può restituire un solo valore di un certo tipo.

Entrambe le funzioni restituiscono solo un valore (intero), ovvero la nuova dimensione oppure un valore di errore (`-1`).

Tuttavia le funzioni agiscono anche per "**side effect**", modificando il contenuto dell'array.

Dal punto di vista pratico:

Le funzioni continuano a restituire un solo valore, ma possiamo anche "esportare" le azioni fatte dalla funzione con il "grimaldello" rappresentato dagli array.

A questo punto acquistano senso anche funzioni che non restituiscono nulla (**procedure**), ma agiscono solo per "**side effect**" modificando lo stato della memoria.

Esiste un modo per riprodurre il comportamento dei vettori, ovvero passare ad una funzione l'indirizzo dell'area di memoria su cui andare a scrivere?