Libreria per la gestione/manipolazione dei vettori

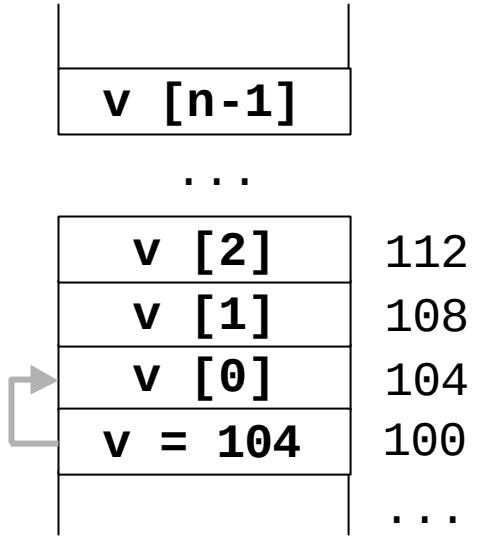
Consideriamo un vettore allocato dinamicamente

```
int * v;
v = (int *) malloc (n * sizeof (int) );
```

"Conversione di tipo" da "void *" *a* "int *"

Numero di elementi

Dimensione di ogni elemento



Noto un vettore (tramite il relativo puntatore) ***v****, non è possibile conoscerne la sua dimensione, cioè il numero di elementi del vettore.*

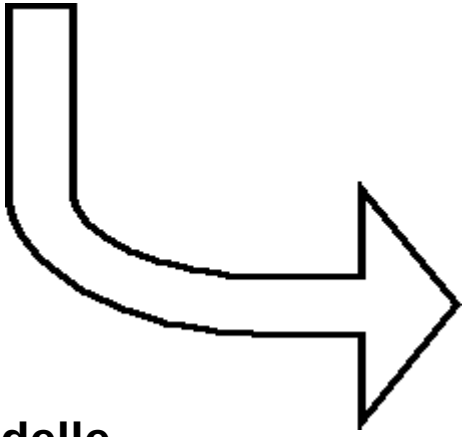
```
int somma (int * vettore)
{
  int i, s = 0;
  for (i = 0;i < ????; i++)
    s += vettore [i];
  return s;
}
```

```
int somma (int * vettore, int dim)
{
  int i, s = 0;
  for (i = 0; i < dim; i++)
    s += vettore [i];
  return s;
}
```

Libreria per la gestione/manipolazione dei vettori

Supponiamo di voler creare una **libreria per la gestione dei vettori dinamici**, cioè un insieme di **funzioni** che possano essere riutilizzate ogni volta sia necessario adoperare vettori dinamici (vettori di interi).
Es.:

creazione di un vettore di dimensioni *n*
sommatoria degli elementi
produttoria degli elementi
calcolo del massimo/minimo
ordinamento degli elementi



Dalle **specifiche**, deriviamo i **prototipi delle funzioni** che ci interessano

```
int * new_vector (int size);
int sum (int * v, int size);
int prod (int * v, int size);
int max (int * v, int size);
int min (int * v, int size);
void sort (int * v, int size);
```

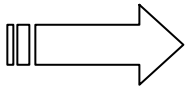
Libreria per la gestione/manipolazione dei vettori

```
int * new_vector (int size);
int sum (int * v, int size);
int prod (int * v, int size);
int max (int * v, int size);
int min (int * v, int size);
void sort (int * v, int size);
```

Ad eccezione di new_vector, tutte le altre funzioni richiedono **sempre**, come parametri, la coppia **puntatore al vettore** (***v***) e **dimensioni del vettore** (***size***).

Poiché **v** e **size** sono sempre legati tra loro, ha senso racchiuderli in una struttura unica:

```
typedef struct {
  int size;
  int * data;
} t_vector;
```



Definiamo dunque un nuovo **tipo** costituito da una struttura contenente il puntatore al vettore ed il numero di elementi di quest'ultimo.

Struttura dati e prototipi finali delle funzioni

```
typedef struct {
  int size;
  int * data;
} t_vector;

t_vector new_vector (int size);
int sum (t_vector v);
int prod (t_vector v);
int max (t_vector v);
int min (t_vector v);
void sort (t_vector v);
```

Libreria per la gestione/manipolazione dei vettori

```
/*
 * libreria funzioni su vettori di interi
 */

#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int size;
  int * data;
} t_vector;


t_vector new_vector (int size)
{
  t_vector vector;
  vector.size = size;
  vector.data = (int *)malloc (sizeof (int) * size);
  return vector;
}
……
```

Libreria per la gestione/manipolazione dei vettori

```
……
int max (t_vector v)
{
  int max_value, i;
  max_value = v.data [0];
  for (i = 1; i < v.size;i++) {
    if (v.data [i] > max_value)
      max_value = v.data [i];
  }
  return max_value;
}

int min (t_vector v)
{
  int min_value, i;
  min_value = v.data [0];
  for (i = 1; i < v.size;i++) {
    if (v.data [i] < min_value)
      min_value = v.data [i];
  }
  return min_value;
}
……
```

Libreria per la gestione/manipolazione dei vettori

```
……
int sum (t_vector v)
{
  int i, sum_value;

  for (i = 0, sum_value = 0; i < v.size;i++)
    sum_value += v.data[i];
  return sum_value;
}


int prod (t_vector v)
{
  int i, prod_value;

  for (i = 0, prod_value = 1; i < v.size;i++)
    prod_value *= v.data[i];
  return prod_value;
}
……
```

Libreria per la gestione/manipolazione dei vettori

```
……
void print_vector (t_vector v)
{
  int i;
  for (i = 0;i < v.size;i++)
    printf ("%d ", v.data[i]);
  printf ("\n");
}

void bubble_sort (t_vector v)
{
  int i, aux;
  char swap_done;

  do {
    swap_done = 0;
    for (i = 0; i < v.size - 1; i++)
      if (v.data [i] > v.data [i + 1]) {
        aux = v.data [i]; v.data [i] = v.data [i + 1];
        v.data [i + 1] = aux; swap_done = 1;
      }
  }
  while (swap_done);
}
……
```

Libreria per la gestione/manipolazione dei vettori

```
/* Main per il test delle funzioni */
int main (int argc, char * argv[])
{
  t_vector myvector;
  int size, i;

  printf ("Inserisci le dimensioni del vettore: ");
  scanf ("%d", &size);
  myvector = new_vector (size);

  for (i = 0; i < size; i++) {
    printf ("Inserisci l'elemento %d:", i);
    scanf ("%d", &myvector.data [i]);
  }
  printf ("Min %d\n", min (myvector));
  printf ("Max %d\n", max (myvector));
  printf ("Sommatoria %d\n", sum (myvector));
  printf ("Produttoria %d\n", prod (myvector));
  bubble_sort (myvector);
  print_vector (myvector);
}
```

```
void bubble_sort (t_vector v)
{
…
}

/* Main per il test delle funzioni */
int main (int argc, char * argv[])
{
  t_vector myvector;
…
  bubble_sort (myvector);
…
}
```

La funzione bubble_sort **modifica** il vettore passato come parametro **myvector**

E allora, perché **myvector** è passato per **valore**? Non dovrebbe essere passato per **riferimento**?

Ricordiamo, tuttavia, che myvector non contiene il vettore vero e proprio, ma un **puntatore** al vettore

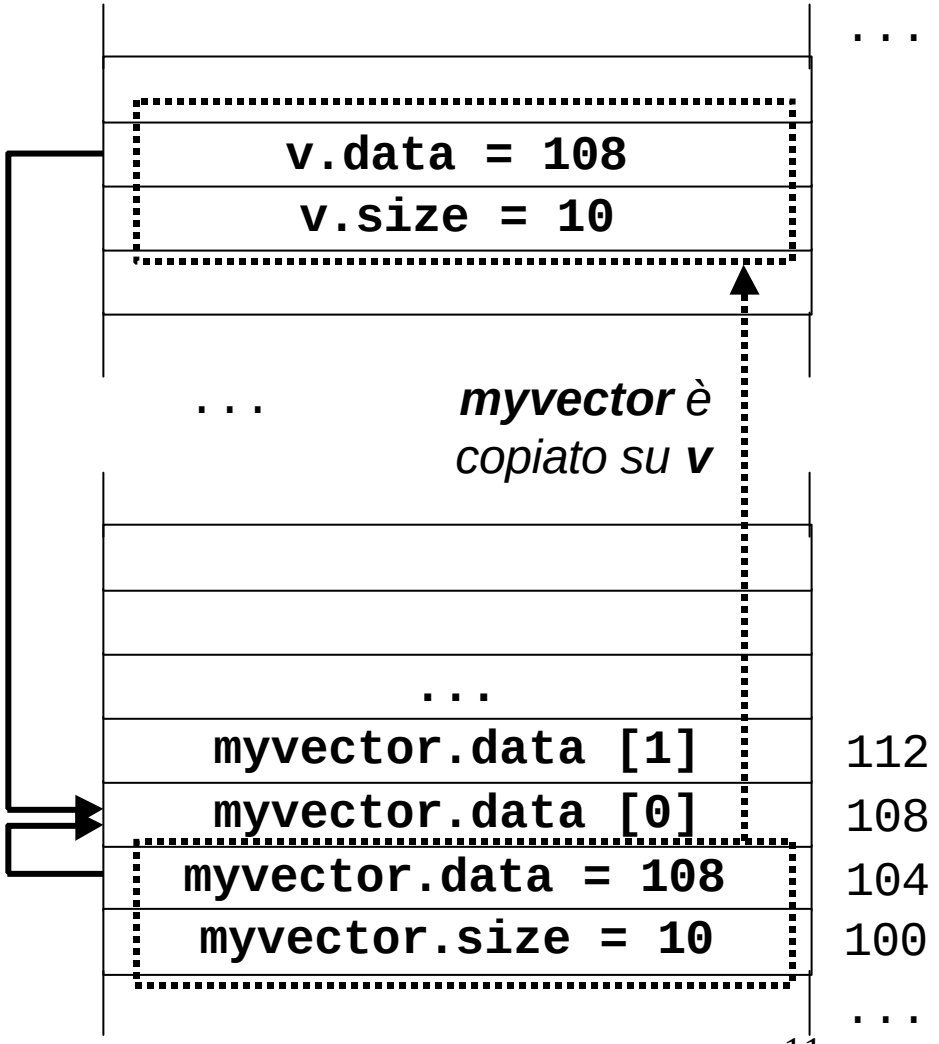
```
typedef struct {
  int size;
  int * data;
} t_vector;
```

Strutture, sort e passaggio per valore

```
typedef struct {
  int size;
  int * data;
} t_vector;

void bubble_sort (t_vector v)
{
…
}
```

```
/* Main per il test delle funzioni */
int main (int argc, char * argv[])
{
  t_vector myvector;
…
  bubble_sort (myvector);
…
}
```



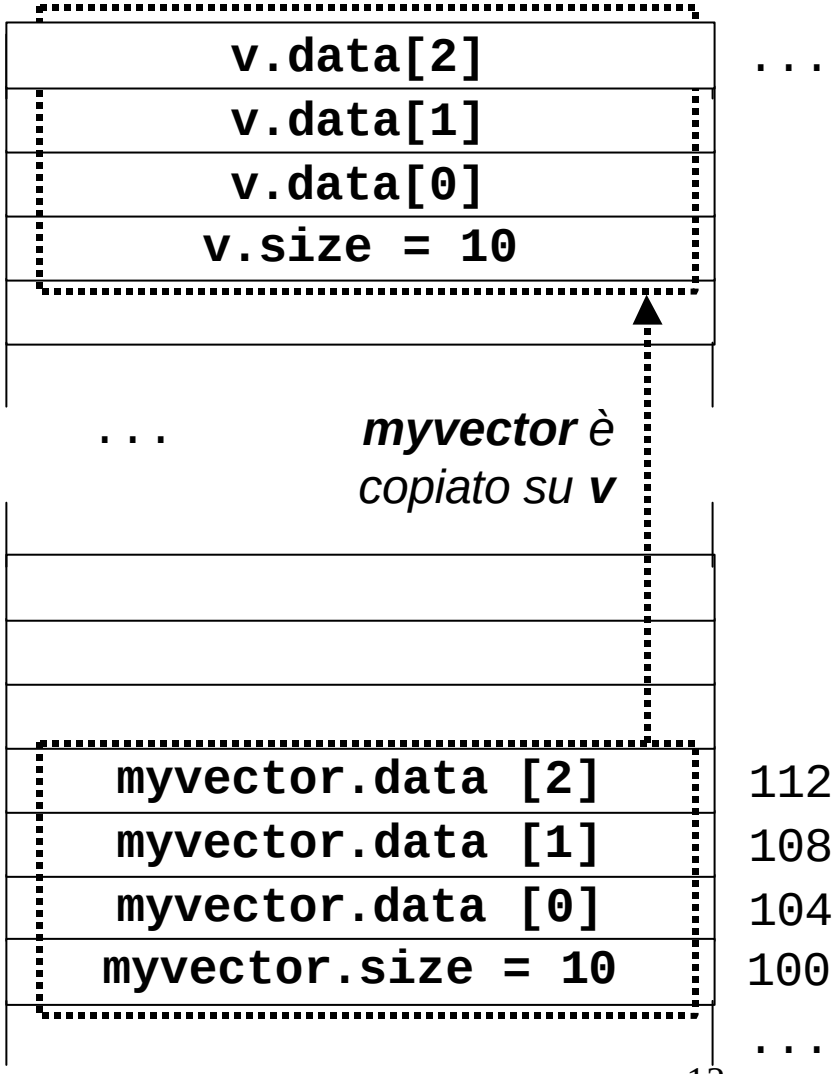
Strutture, sort e passaggio per valore **con vettore statico**

```
typedef struct {
  int size;
  int data[3];
} t_vector;

void bubble_sort (t_vector v)
{
…
}
```

In questo caso, le modifiche fatte da `bubble_sort` su ***v*** **non** si riflettono su ***myvector***.

```
/* Main per il test delle funzioni */
int main (int argc, char * argv[])
{
  t_vector myvector;
…
  bubble_sort (myvector);
…
}
```



Liberare la memoria allocata

L'allocazione dinamica (`malloc`) ha il vantaggio di consentire di liberare la memoria allocata, quando i dati non servono più.

Per liberare la memoria allocata si usa la funzione:

```
void free (void * ptr);
```

Completiamo la nostra libreria dei vettore con la funzione che **distrugge** un vettore precedentemente allocato:

```
void delete_vector (t_vector v)
{
  free (v.data);
}
```

“Homework”: il minimo comune multiplo

Arricchiamo la libreria di gestione dei vettori con una funzione che calcola il minimo comune multiplo dei valore presenti nel vettore:

```
int mcm (t_vector v);
```

mcm (n_1,n_2,…,n_k)

i_1 = mcm (n_1, n_2)
i_2 = mcm (i_1, n_3)
i_3 = mcm (i_2, n_4)
…
Ris = mcm (i_{k-2}, n_k)