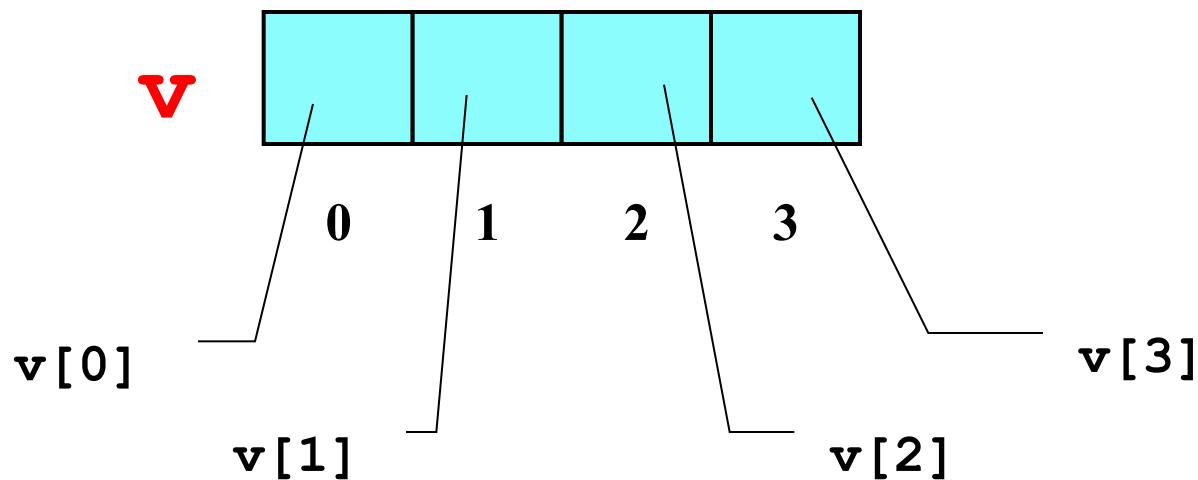


ARRAY: STRUTTURA FISICA

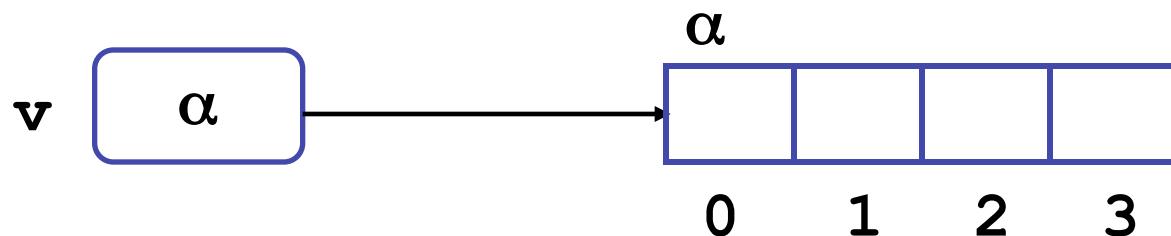
Un *array* è una collezione finita di N variabili dello stesso tipo, ognuna identificata da un indice compreso fra 0 e $N-1$



Praticamente, le cose non stanno proprio così.

ARRAY: STRUTTURA FISICA

- In C un *array* è in realtà un puntatore che punta a un'area di memoria pre-allocata, di dimensione prefissata.



Pertanto, *il nome dell'array è un sinonimo per il suo indirizzo iniziale:* $v \equiv \&v[0] \equiv \alpha$

CONSEGUENZA

- Il fatto che il nome dell' array non indichi l' array, ma l' indirizzo iniziale dell' area di memoria ad esso associata ha una conseguenza:

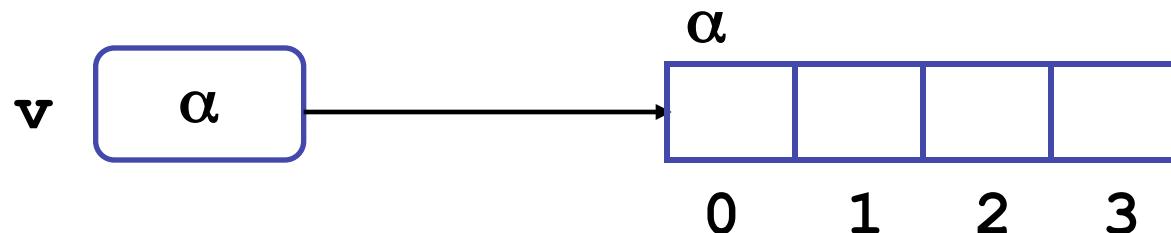
È impossibile denotare un array nella sua globalità, in qualunque contesto.

- Quindi non e' possibile:
 - assegnare un array a un altro (**s2 = s**)
 - che una funzione restituisca un array
 - passare un array come parametro a una funzione
non significa affatto passare l'intero array !!

ARRAY PASSATI COME PARAMETRI

Poiché un *array* in C è un puntatore che punta a un'area di memoria pre-allocata, di dimensione prefissata, il nome dell'array:

- non rappresenta l'intero array
- è un alias per il suo indirizzo iniziale
 $(v \equiv \&v[0] \equiv \alpha)$



ARRAY PASSATI COME PARAMETRI

Quindi, *passando un array a una funzione:*

- *non si passa l'intero array !!*
- si passa solo (per valore!) il suo indirizzo iniziale

$$(\mathbf{v} \equiv \&\mathbf{v}[0] \equiv \alpha)$$

- *agli occhi dell'utente, l'effetto finale è che l'array e' passato per riferimento!!*

CONCLUSIONE

A livello fisico:

- il C passa i parametri *sempre e solo per valore*
- nel caso di un array, si passa il suo indirizzo iniziale ($v \equiv \&v[0] \equiv \alpha$) perché *tale è il significato del nome dell'array*

A livello concettuale:

- il C passa *per valore* tutto tranne gli array, che vengono trasferiti *per riferimento*.

ESEMPIO

Problema:

Scrivere una funzione che, dato un array di N interi, ne calcoli il massimo.

Si tratta di riprendere l' esercizio già svolto, e impostare la soluzione come funzione anziché codificarla direttamente nel *main*.

Dichiarazione della funzione:

```
int findMax(int v[], int dim);
```

ESEMPIO

Il cliente:

```
int main() {  
    int max, v[] = {43,12,7,86};  
    max = findMax(v, 4);  
    return 0;}
```



Trasferire esplicitamente la dimensione dell' array è NECESSARIO, in quanto la funzione, ricevendo solo l'indirizzo iniziale, non avrebbe modo di sapere quanto è lungo l'array !

ESEMPIO

La funzione:

```
int findMax(int v[], int dim) {  
    int i, max;  
    max=v[0];  
    for (i=1; i<dim; i++)  
        if (v[i]>max) max=v[i];  
    return max;  
}
```

ESEMPIO

La funzione:

Per evitare che la funzione modifichi l' array (visto che è passato per riferimento), si può imporre la qualifica **const**

Se lo si tenta: *cannot modify a const object*



```
int findMax(const int v[], int dim) {  
    int i, max;  
    max=v[0];  
    for (i=1; i<dim; i++)  
        if (v[i]>max) max=v[i];  
    return max;  
}
```

ESERCIZIO: Max e Min di un vettore

```
int minimo (int vet[], int Dim)
{int i, min;
min = vet[0];
for (i = 1; i < Dim; i++)
    if (vet[i]<min)
        min = vet[i];
return min;
}
```

```
int massimo (int vet[], int Dim)
{int i, max;
max = vet[0];
for (i = 1; i < Dim; i++)
    if (vet[i]>max)
        max=vet[i];
return max;
}
```

ESERCIZIO: MAX E MIN DI UN VETTORE

```
#define N 15
```

```
int main ()
{int i, a[N];
printf ("Scrivi %d numeri interi\n", N);
for (i = 0; i < N; i++)
    scanf ("%d", &a[i]);
printf ("L'insieme dei numeri è: ");
for (i = 0; i<N; i++)
    printf("%d",a[i]);
printf ("Il minimo vale %d e il
massimo è %d\n", minimo(a,N),
massimo(a,N));
return 0;}
```

```
#define N 15

int main ()
{int i, a[N];
printf ("Scrivi %d numeri interi\n", N);
leggi(a,N);
printf ("L'insieme dei numeri è: ");
stampa(a,N);
printf ("Il minimo vale %d e il
massimo è %d\n", minimo(a,N),
massimo(a,N));
return 0; }
```

ESERCIZIO: RICERCA DI UN ELEMENTO

```
int ricerca (int vet[], int el, int Dim)
{int i=0;
 int T=0;
 while ((i<Dim) && (T==0))
 { if (el==vet[i]) T=1;
   i++; }
 return T;
}
```

ESERCIZIO: RICERCA DI UN ELEMENTO

```
#include <stdio.h>
#define N 15

int main ()
{int i, num;
 int a[N];
printf ("Scrivi %d numeri interi\n", N);
for (i = 0; i < N; i++)
    scanf ("%d", &a[i]);
printf ("Valore da cercare: ");
scanf ("%d", &num);
if (ricerca(a, num, N)) printf ("\nTrovato\n");
    else printf ("\nNon trovato\n");
return 0;}
```

ESERCIZIO: RICERCA DI UN ELEMENTO

- Sapendo che il vettore è ***ordinato***, la ricerca può essere ottimizzata.

- ***Vettore ordinato in senso non decrescente:***

- Esiste una relazione d'ordine totale sul dominio degli elementi del vettore e:
 - $i < j$ si ha $v[i] \leq v[j]$

2	3	5	5	7	8	10	11
---	---	---	---	---	---	----	----

- ***Vettore ordinato in senso crescente:***

- $i < j$ si ha $v[i] < v[j]$

2	3	5	6	7	8	10	11
---	---	---	---	---	---	----	----

- In modo analogo si definiscono l'ordinamento in senso ***non crescente*** e ***decrescente***.

ESERCIZIO: RICERCA BINARIA

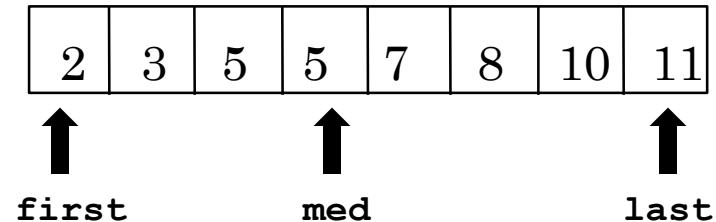
- Ricerca binaria di un elemento in un vettore ordinato in senso non decrescente in cui il primo elemento è **first** e l'ultimo **last**.
- La tecnica di ***ricerca binaria*** rispetto alla ricerca esaustiva, consente di eliminare ad ogni passo metà degli elementi del vettore.

ESERCIZIO: RICERCA BINARIA

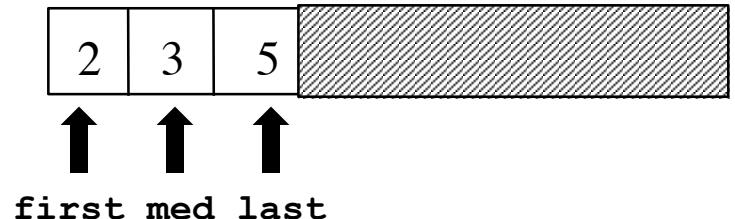
- Si confronta l' elemento cercato **e1** con quello mediano del vettore, **v [med]** .
- Se **e1 ==v [med]** , fine della ricerca (**trovato=true**).
- Altrimenti,
 - se il vettore ha almeno due componenti (**first < last**):
 - se **e1<v [med]** , ripeti la ricerca nella prima metà del vettore (indici da **first** a **med-1**);
 - se **e1>v [med]** , ripeti la ricerca nella seconda metà del vettore (indici da **med+1** a **last**).

ESERCIZIO: RICERCA BINARIA

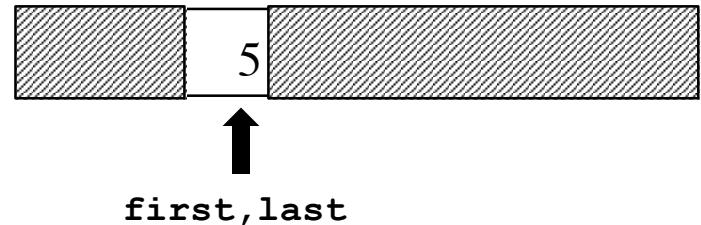
- Esempio: si cerca il valore $el=4$
- $med = (first+last)/2$
- $el < v[med]$



- $el > v[med]$



- Vettore a una componente:
fine della ricerca con
insuccesso



ESERCIZIO: RICERCA BINARIA

```
int ricerca_bin (int vet[], int first, int last,
                  int el)
{int med=(first+last)/2;
 int T=0;
 while ((first<=last) && (T==0))
 {
     if (el==vet[med]) T=1;
     else if (el < vet[med]) last=med-1;
         else first=med+1;
     med = (first + last) / 2;
 }
return T;
}
```

ESERCIZIO: Ricerca di un elemento

```
#include <stdio.h>
#define N 15

int main ()
{int i,num;
 int a[N];
 printf ("Scrivi %d numeri interi ordinati\n", N);
 for (i = 0; i < N; i++)
     scanf ("%d", &a[i]);
 printf ("Valore da cercare: ");
 scanf ("%d", &num);
 if (ricerca_bin(a,0,N,num))
     printf ("\nTrovato\n");
     else printf ("\nNon trovato\n");
 return 0;
}
```

RICERCA BINARIA RICORSIVA

```
int ricerca_bin (int vet[], int first, int last, int el)
{ int med=(first + last)/2;
  if (first > last)
    return 0;
  else
    if (el==vet[med])  return 1;
    else
      if (el > vet[med])
        return ricerca_bin(vet, med+1, last, el);
      else
        return  ricerca_bin(vet, first, med-1, el);
}
```

ESERCIZIO: Ricerca di un elemento

```
#include <stdio.h>
#define N 7

int main()
{int i,num;
 int a[N];
printf ("Scrivi %d numeri interi ordinati\n", N);
for (i = 0; i < N; i++)
    scanf("%d", &a[i]);
printf ("Valore da cercare: ");
scanf ("%d",&num);
if (ricerca_bin(a,0,N,num)==1)
    printf("\nTrovato\n");
    else printf("\nNon trovato\n");
return 0;}
```