## TIPI DI DATO

Un **tipo di dato** T è definito come:

- un **dominio di valori, D**
- un **insieme di funzioni** $F_1,..,F_n$ sul dominio D
- un **insieme di predicati** $P_1,..,P_m$ sul dominio D

$$T = \{ D, \{F_1,...,F_n\}, \{P_1,...,P_m\} \}$$

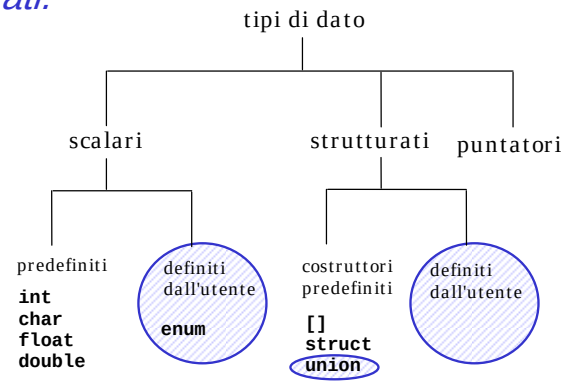
## TIPI DI DATO: ESEMPIO

Il **tipo di dato** INTERO è definito come:

- un **dominio di valori, Z**
- un **insieme di funzioni** $F_1,..,F_n$ sul dominio D
  - esempio SOMMA, SOTTRAZIONE, PRODOTTO ….
- un **insieme di predicati** $P_1,..,P_m$ sul dominio D
  - ad esempio MAGGIORE, MINORE, UGUALE...

## TIPI DI DATO

I tipi di dato si differenziano in *scalari* e *strutturati.*




Non saranno trattati nel corso

## TIPI DI DATO

In C si possono *definire tipi strutturati.*

Vi sono due *costruttori* fondamentali:

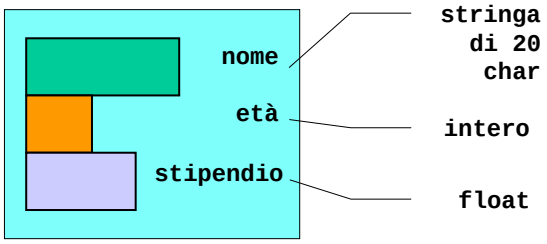
**`[ ]`** *(array)*

**`struct`** *(strutture)*

## STRUTTURE

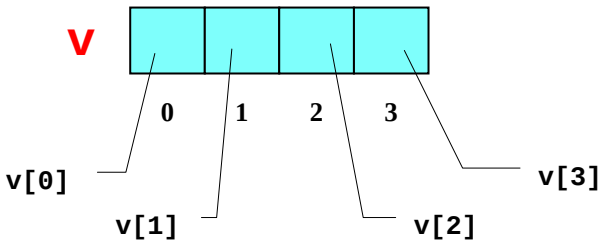
Una *struttura* è una *collezione finita di variabili non necessariamente dello stesso tipo*, ognuna identificata da un *nome*.

**struct persona**




## ARRAY (VETTORI)

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



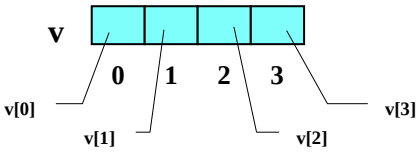
## ARRAY (VETTORI)

**Definizione di una *variabile* di tipo *array*:**

```
<tipo> <nomeArray> [ <costante> ];
```

**Esempi:**

```
int  v[4];
char nome[20];
```




**ATTENZIONE: Sbagliato !!**

~~`int  N;`~~
~~`char nome[N];`~~

Il compilatore non sa come dimensionare l'array

## ESEMPIO

- **Problema: leggere da tastiera gli elementi di un vettore**

```
#include <stdio.h>
#define N 3

void main()
{ int k;
  int A[N];

for(k=0; k < N; k++)
  {printf("Dammi elemento %d: ", k);
   scanf("%d", &A[k]);
  }
}
```

**Direttiva gestita dal preprocessore: sostituzione di testo**

## ESEMPIO

- **<u>Problema</u>: inizializzare un vettore con il prodotto degli indici**

```
#include <stdio.h>
#define N 3

void main()
{ int i=0;
  int A[N];

while (i<N)
  {
    A[i]=i*i;   /*gli elementi del vettore sono 0,1,4*/
   i++;
  }
}
```

## ESEMPIO

- **<u>Problema</u>: scrivere un programma che, dato un vettore di N interi, determini il valore massimo.**

**<u>Specifica di I livello:</u>**

Inizialmente, si assuma come *massimo di tentativo* il primo elemento. $m_0 = v[0] \rightarrow m_0 \geq v[0]$

Poi, si confronti via via il massimo di tentativo con gli elementi del vettore: nel caso se ne trovi uno maggiore del massimo di tentativo attuale, si aggiorni il valore del massimo.

$m_i = max(m_{i-1}, v[i]) \rightarrow m_i \geq v[0], v[1]..v[i]$

Al termine, il valore del massimo di tentativo coincide col valore massimo ospitato nel vettore. $m_{n-1} \geq v[0], v[1] \ldots v[n-1]$ cioè $m_{n-1}$ è il max cercato.

## ESEMPIO

**Codifica:**

```
#define DIM 10
void main() {
  int v[DIM]; int i, max;

  for(i=0; i < N; i++)     /* INIZIO LETTURA */
  {printf("Dammi elemento %d: ", i);
  scanf("%d", &v[i]);}     /* FINE LETTURA */

  max=v[0];
  for (i=1; i<DIM; i++)
   if (v[i]>max) max = v[i];
  /* ora max contiene il massimo */
 printf("Massimo = %d", max);
}
```

## DIMENSIONE FISICA VS. LOGICA

- Un array è una collezione finita di N celle dello stesso tipo
- Questo non significa che si debbano per forza *usare sempre tutte!*
- La *dimensione logica* di un array può essere inferiore (mai superiore!) alla sua *dimensione fisica*
- Spesso, la *porzione di array* realmente utilizzata *dipende dai dati d'ingresso.*

## DIMENSIONE FISICA VS. LOGICA

**Esempio**

È data una serie di rilevazioni di temperature espresse in gradi Kelvin.
**Ogni serie è composta di <u>al più 10 valori</u>,** ***ma può essere più corta***. Il valore "-1" indica che la serie delle temperature è finita.

Scrivere un programma che, data una serie di temperature, calcoli la media delle temperature fornite.

## ESEMPIO

- Il vettore deve essere *dimensionato per 10 celle* (caso peggiore)...
- … ma la porzione realmente usata *può essere minore!*

**<u>Specifica di I livello:</u>**

- leggere le temperature e memorizzarle nel vettore
- calcolare la somma di tutti gli elementi del vettore, e nel frattempo contare quanti sono
- il risultato è il rapporto fra la somma degli elementi così calcolata e il numero degli elementi.

## ESEMPIO

**Specifica di II livello:**

Leggi gli elementi del vettore finche' i e' minore della dimensione massima e l'elemento letto non e' -1

Al termine (quando o un elemento vale -1, oppure hai esaminato N elementi), l'indice i rappresenta il numero totale di elementi ossia la dimensione LOGICA del vettore.

## ESEMPIO

**Specifica di II livello (continua):**

Inizialmente, poni uguale a 0 una variabile S che rappresenti la somma corrente

```
s = 0
```

A ogni passo (da 0 a i), aggiungi l'elemento corrente a una variabile S che funga da somma.

```
s = s + v[k],
```

Al termine (dopo i elementi), si ottiene il valore finale della somma: il risultato è il rapporto S/k.

## ESEMPIO

```
#define DIM 10
void main() {
  int k, v[DIM], i = 0;
  float media, s=0;
  printf("inserisci temp. - 1 per terminare");
  scanf("%d", &v[0]);
  while ((v[i]!= -1) && (i < DIM - 1))
      {i++;
      printf("inserisci temp.");
      scanf(" %d", &v[i]);          }
  for (k=0; k < i ; k++)
      s = s + v[k];
  media = s / i;
  printf("Media = %f", media);
}
```

Dimensione fisica = 10

Dimensione logica = i

## INPUT OUTPUT

- Non e' possibile leggere/scrivere un intero vettore (a parte come vedremo le ***stringhe***); occorre leggere/scrivere le sue componenti:

```
void main() {
  int i,frequenza[25];
  for (i=0; i<25; i++)
  {   scanf("%d",&frequenza[i]);
      frequenza[i]=frequenza[i]+1;
  }   /*    legge a terminale le componenti del
            vettore frequenza e le incrementa
      */
}
```

## ASSEGNAMENTO

- Anche se due variabili vettore sono dello ***stesso tipo***, non e' possibile l'assegnamento diretto:

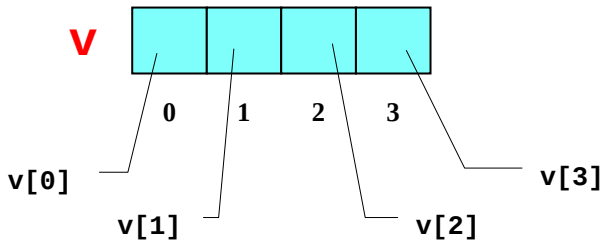
```
int F[25], frequenza[25];
F=frequenza;          /* NO */
```

- ma occorre copiare componente per componente:

```
for (i=0; i<25; i++)
     F[i]=frequenza[i];
```

## ARRAY: STRUTTURA FISICA

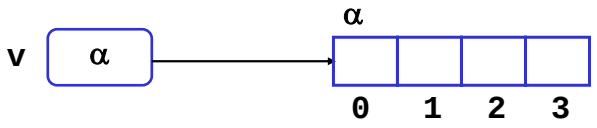
Un *array* è una *collezione finita di N variabili dello stesso tipo*, ognuna identificata da un *indice* compreso fra 0 e N-1




Praticamente, le cose non stanno proprio così.

## ARRAY: STRUTTURA FISICA

- In C un *array* è in realtà un <u>puntatore</u> 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` ≡ `&v[0]` ≡ α

## 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:

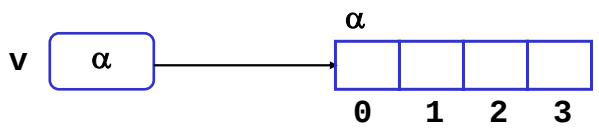
  *È impossibile denotare un array nella sua globalità, in qualunque contesto.*

- 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* <u>*non significa affatto passare l'intero array !!*</u>

## 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`** ≡ **`&v[0]`** ≡ α)




## 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
  (**`v`** ≡ **`&v[0]`** ≡ α)

- *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 ≡ &v[0]`** ≡ α) *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:**

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

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


void 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));
}
```

## 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


void 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");
}
```

## 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]<=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 e' **`first`** e l'ultimo **`last`**.

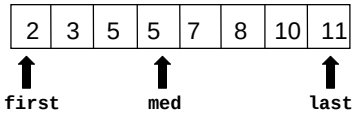
- La tecnica di ***ricerca binaria*** rispetto alla ricerca esaustiva, consente di eliminare ad ogni passo metà degli elementi del vettore.

## ESERCIZIO: RICERCA BINARIA

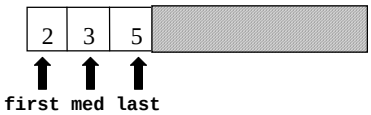
- Si confronta l'elemento cercato **`el`** con quello mediano del vettore, **`V[`*`med`*`]`**.
- Se **`el ==V[med]`**, fine della ricerca (**`trovato=true`**).
- Altrimenti,
- se il vettore ha almeno due componenti (**`first < last`**):
  - se **`el<V[med]`**, ripeti la ricerca nella prima metà del vettore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

## ESERCIZIO: RICERCA BINARIA

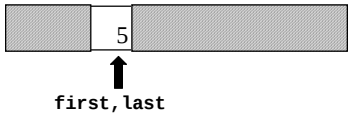
- Esempio: si cerca il valore **`el=4`**
- **`med = (first+last)/2`**
- **`el < V[med]`**




- **`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

void 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");
}
```

## 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


void 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");
}
```