

TIPI DI DATO

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

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

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

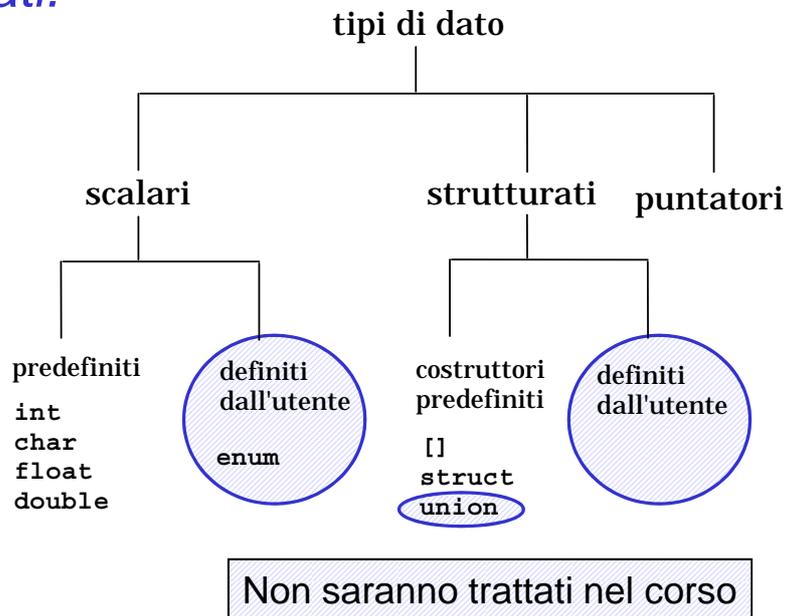
TIPI DI DATO: ESEMPIO

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

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

TIPI DI DATO

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



TIPI DI DATO

In C si possono *definire tipi strutturati*.

Vi sono due *costruttori* fondamentali:

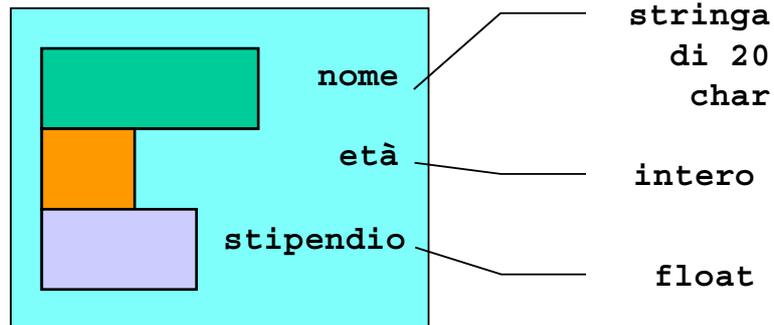
[] (array)

struct (strutture)

STRUTTURE

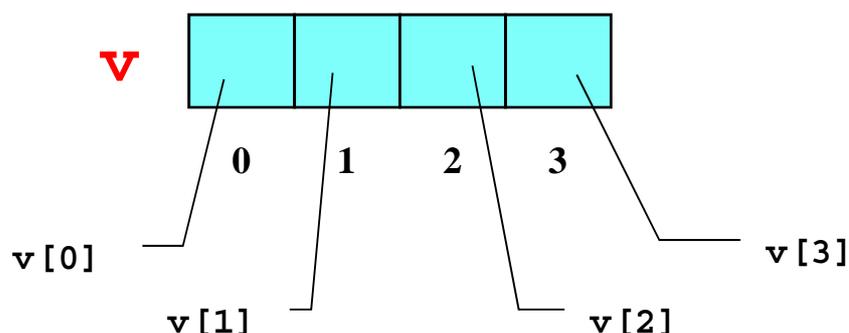
Una *struttura* è una collezione finita di variabili non necessariamente dello stesso tipo, ognuna identificata da un *nome*.

`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



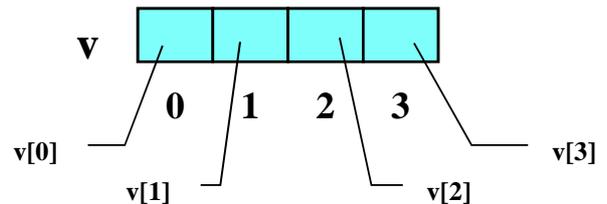
ARRAY (VETTORI)

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
```



Direttiva gestita dal preprocessore:
sostituzione di testo

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

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

ESEMPIO

- **Problema:** inizializzare un vettore con il prodotto degli indici

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

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

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

ESEMPIO

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

Specifica di I livello:

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.

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

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

ESEMPIO

Codifica:

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

    for(i=0; i < DIM; 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);
    return 0;
}
```

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 al più 10 valori, *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!*

Specifica di I livello:

- 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 finché i è minore della dimensione massima e l'elemento letto non è -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 → Dimensione fisica = 10
int 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++; → Dimensione logica = 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);
    return 0; }
```

INPUT OUTPUT

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

```
int 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
          */
    return 0;
}
```

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];
```