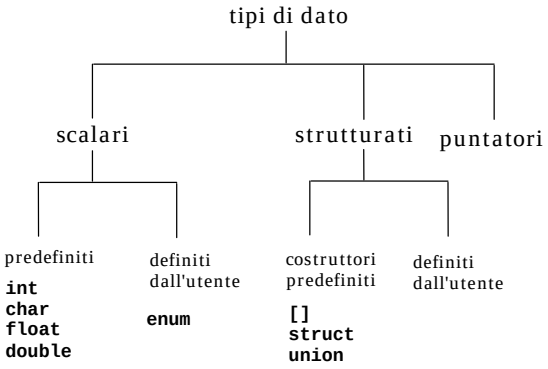
TIPI DI DATO

Un **tipo di dato** T (o tipo di dato astratto) 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

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



In C si possono definire nuovi tipi strutturati.

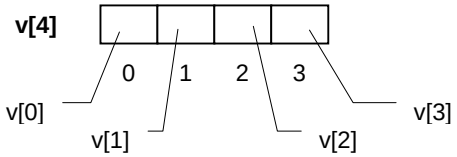
TIPI STRUTTURATI - COSTRUTTORI PREDEFINITI

Il linguaggio C fornisce due *costruttori* fondamentali:

- **`[ ]`** ***(vettori)***
- **`struct`** ***(strutture)***

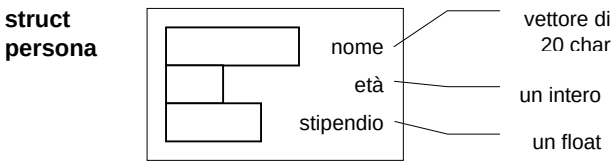
VETTORI

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



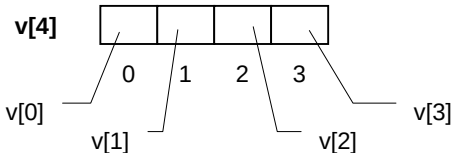
STRUTTURE

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



VETTORI

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



SINTASSI di una *definizione* di una *variabile* di tipo vettore:

<tipo> <identificatore> [<espr-costante>] ;

ESEMPIO:

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

SINTASSI di una *definizione* di un *tipo vettore*:

`typedef <tipo-base> <nome-tipo> [ <dimensione> ];`

ESEMPIO

```
typedef int vett4[4];
typedef char stringa [20];
```

Uso:

```
vett4   v;
stringa s;
```


ESEMPIO 1

Scrivere un programma che, dato un vettore di N interi, determini il valore massimo.

SPECIFICA

< Assumi come massimo di tentativo il primo elemento. Poi, confronta via via il massimo di tentativo con gli elementi del vettore, e se ne trovi uno maggiore, aggiorna il massimo >

SPECIFICA di II° livello

```
max = v[0];  indice = 1;
while (i < N)
    < se max<v[i], aggiorna max >
```

CODIFICA

```
main(){
  const int N = 4;      /* espressione costante */
  int max, i, v[N] = {43,12,7,86};
  for (max = v[0], i=1; i<N; i++)
        if (max < v[i])  max = v[i];
}
```

NOTE:

- la definizione di `v` è corretta perché `N` è una costante. Se fosse una variabile, si avrebbe un errore.
- l'espressione `v[N] = {43,12,7,86}` inizializza il vettore `v` con i valori specificati.

ESEMPIO 1 (segue)

UNA CODIFICA ALTERNATIVA

Anziché inizializzare il vettore con dei valori prestabiliti, è possibile inserire nelle sue *celle* dei valori calcolati.

Ad esempio, se si vuole inizializzare il vettore a {1,2,3,4} si potrebbe scrivere il seguente *ciclo di inizializzazione*:

```
main(){
  const int N = 4;       /* espressione costante */
  int max, i, v[N];      /* NESSUNA INIZIALIZZ. */
  for (i=0; i<N; i++) v[i] = i+1;  /* {1,2,3,4} */
  for (max = v[0], i=1; i<N; i++)
       if (max < v[i])  max = v[i];
}
```

NOTE:

- **nel solo caso in cui v sia inizializzato esplicitamente** la dimensione N può essere omessa, in quanto deducibile dal numero di elementi elencati nella lista di inizializzazione:

```
int max, i, v[] = {43,12,7,86};
```

OSSERVAZIONE

Contrariamente ad altri linguaggi, il C non consente di scegliere il valore iniziale dell'indice, che è sempre 0.

Quindi, un vettore di N elementi ha sempre, necessariamente, indici da 0 a N-1 (inclusi).

VETTORI - RIFLESSIONI

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

Questo non significa che si debbano per forza *usare* tutte le celle disponibili!
Spesso, la porzione di vettore realmente utilizzata *dipende dai dati di ingresso*.

Si consideri ad esempio il seguente problema:

È data una serie di rilevazioni di temperature espresse in gradi Kelvin.

Ogni serie è composta di **al massimo 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 memorizzata in un vettore, calcoli la media delle temperature fornite.

ESEMPI di serie di temperature possibili:

273, 340, 45 [-1]

273 [-1]

243, 567, 300, 4500, 678, 256, 500, 340, 650, 600

Il vettore dovrà essere **dimensionato a 10 celle**, perché le temperature possono essere fino a 10, **ma le celle realmente utilizzate potranno essere meno**.

ESEMPIO 2

È dato un vettore di **al più** 10 interi non negativi, che rappresentano temperature espresse in gradi Kelvin.

Le temperature realmente disponibili, però possono essere anche **meno di 10**: il valore "-1" indica che la serie delle temperature è finita.

Scrivere un programma che, dato tale vettore di temperature, calcoli la media delle temperature fornite.

SPECIFICA

< Somma tutti gli elementi del vettore e contali. Calcola poi il rapporto fra tale somma e il numero di elementi>

SPECIFICA di II° livello

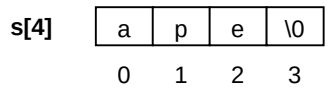
< Scandisci gli elementi del vettore fino a che o incontri un -1, oppure hai scandito tutti gli N elementi. Aggiungi ogni elemento a una variabile S che funga da somma. Al termine, l'indice K a cui sei giunto rappresenta il numero di elementi: calcola dunque il rapporto S/K.>

CODIFICA

```
main(){
  const int N = 10;      /* espressione costante */
  int media, s=0, k, v[10] = {273,340,467,-1};
  for (k=0; k<N && v[k]>=0; k++) s += v[k];
  media = s / k;
}
```

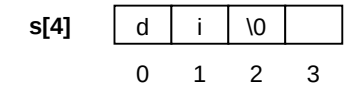

STRINGHE DI CARATTERI

Una stringa di caratteri in C è un vettore di caratteri, terminato dal carattere '\0' (il carattere di codice ASCII 0).



Un vettore di N caratteri può dunque ospitare stringhe lunghe ***al più N-1 caratteri***, in quanto una cella deve essere destinata al carattere di terminazione.

Ovviamente, un vettore di N caratteri può ben essere usato per memorizzare stringhe *più corte:*



In questo caso *le celle oltre la k-esima* (*k* essendo la lunghezza della stringa) *sono "concettualmente vuote",* cioè praticamente inutilizzate e contenenti un valore casuale.

INIZIALIZZAZIONE DI STRINGHE

Una stringa si può inizializzare come ogni altro vettore:

```
char s[4] = {'a','p','e','\0'};
```

oppure anche così:

```
char s[4] = "ape";   /* SOLO STRINGHE */
```

ESEMPIO 3

Scrivere un programma che, data una stringa di caratteri, ne calcoli la lunghezza.

SPECIFICA

Ipotesi: la stringa fornita è "ben formata", ossia certamente terminata dal carattere '\0'.

< Scandisci tutti gli elementi del vettore fino a trovare il carattere di fine-stringa ('\0'). L'indice di tale carattere rappresenta la lunghezza della stringa>

CODIFICA

```
main(){
  char s[] = "Nel mezzo del cammin di nostra vita";
  int i;
  for (i=0; s[i]!='\0'; i++) ;
  /* i rappresenta la lunghezza della stringa */
}
```

NOTE:

- il carattere di terminazione è implicito nell'uso di costanti stringa (notazione "sequenza di caratteri fra virgolette")
- non avendo specificato la dimensione, il vettore s è dimensionato esattamente a N+1 caratteri, essendo N la lunghezza della stringa di inizializzazione indicata.

ESEMPIO 4

Scrivere un programma che, data una stringa di caratteri, la copi in un altro vettore di caratteri dato.

SPECIFICA

Ipotesi: la stringa fornita è "ben formata", ossia certamente terminata dal carattere '\0'.
< Scandisci tutti gli elementi del vettore fino a trovare il carattere di fine-stringa ('\0'): mentre scandisci, copia ogni carattere nella corrispondente posizione nel nuovo vettore>

CODIFICA

```
main(){
  char s[] = "Nel mezzo del cammin di nostra vita";
  char s2[40];
  int i;
  for (i=0; s[i]!='\0'; i++) s2[i]=s[i];
  s2[i] = '\0';
}
```

NOTE:

- *il carattere di terminazione deve essere inserito esplicitamente* nella stringa `s2`, perché nel caso in cui `s[i]=='\0'` il ciclo termina e *non effettua* l'assegnamento `s2[i]=s[i]`.
- **cosa succede se s2 è troppo corto???**

Esercizio: provare cosa succede se s2 è troppo corto...

ESEMPIO 4 bis

Scrivere un programma che, data una stringa di caratteri, la copi in un altro vettore di caratteri dato.

SPECIFICA

Ipotesi: la stringa fornita è "ben formata", ossia certamente terminata dal carattere '\0'.
Se il vettore di destinazione non è sufficientemente lungo, la stringa verrà troncata.
< Scandisci tutti gli elementi del vettore fino a trovare il carattere di fine-stringa ('\0') **o fino a quando arrivi alla lunghezza massima del vettore di destinazione (-1)**: mentre scandisci, copia ogni carattere nella cella corrisp.>

CODIFICA

```
main(){
  const int N=30;
  char s[] = "Nel mezzo del cammin di nostra vita";
  char s2[N];
  int i;
  for (i=0; s[i]!='\0' && i<N-1; i++) s2[i]=s[i];
  s2[i] = '\0';
}
```

NOTE:

- il ciclo deve terminare quando `i<N-1` (non `i<N`) in quanto deve comunque esserci lo spazio per il carattere di terminazione `'\0'`.

ESEMPIO 5

Scrivere un programma che, date *due* stringhe di caratteri, verifichi quale precede l'altra in ordine alfabetico.

RAPPRESENTAZIONE DEL RISULTATO

Il risultato può essere di tre tipi: s1<s2, s1==s2, s1>s2. Perciò un boolean non basta. Alternative possibili:
– usare due boolean (`uguale` e `precede`)
– usare tre boolean (`uguale`, `s1PrecedeS2`, `s2PrecedeS1`)
– usare un intero (negativo, 0, positivo).

Scegliamo la terza via.

SPECIFICA

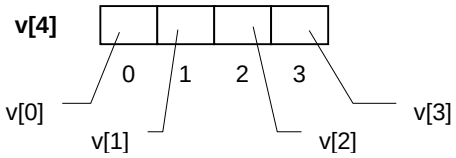
< Scandisci uno per uno gli elementi di egual posizione delle due stringhe, fino a che ne trovi due diversi (se ne trovi). Confronta poi i due caratteri corrispondenti e determina quale è il minore.>

CODIFICA

```
main(){
  char s1[] = "Sempre caro mi fu quest'ermo colle";
  char s2[] = "Sempre odiai quell'orrido colle";
  int i, stato;
  for (i=0;
       s1[i]!='\0' && s2[i]!='\0' && s1[i]==s2[i] ;
       i++) ;
  stato = s1[i]-s2[i];  /* negativo -->  s1 < s2 */
                        /* positivo -->  s1 > s2 */
                        /* zero     -->  s1 == s2 */
}
```

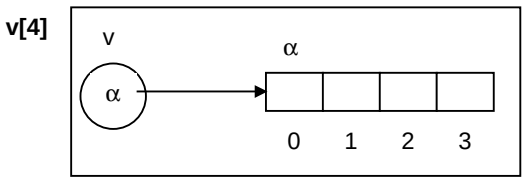
I VETTORI VISTI DA VICINO

Concettualmente, un vettore è una collezione finita di N variabili dello stesso tipo, ognuna identificata da un *indice*:



Praticamente, in C le cose non stanno esattamente così.

In C, un vettore è in realtà organizzato come un *puntatore costante,* inizializzato a puntare a un'area di memoria opportunamente pre-allocata:



Pertanto,

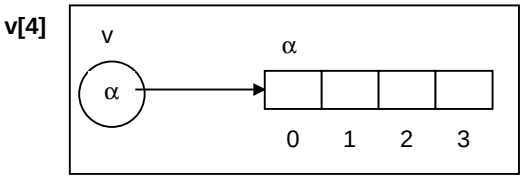
***il nome del vettore*, v, *rappresenta in realtà l'indirizzo iniziale* dell'area di memoria associata:**

`v ≡ & v[0]`

I VETTORI VISTI DA VICINO (II)

Un vettore C è organizzato come un *puntatore costante,* inizializzato a puntare a un'area di memoria pre-allocata

Il nome del vettore rappresenta *l'indirizzo iniziale* dell'area di memoria associata, e vale l'identità `v ≡ & v[0]`.



Il fatto che il nome del vettore indichi *non il vettore in sé*, ma *l'indirizzo iniziale dell'area di memoria* a lui associata ha una importantissima conseguenza:

in C non esiste modo di "indicare" un vettore come un tutt'uno
↓
è impossibile operare su vettori intesi nella loro globalità.

CONSEGUENZE

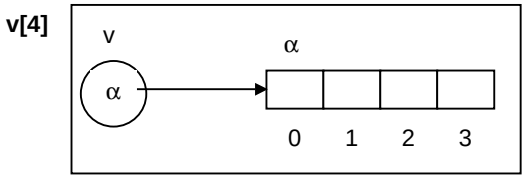
- è impossibile assegnare un vettore a un altro (v1 = v2)
- è impossibile che una funzione restituisca un vettore
- passare un vettore come parametro a una funzione *non significa passare l'intero vettore!*

VETTORI PASSATI COME PARAMETRI

Un vettore C è organizzato come un *puntatore costante,* inizializzato a puntare a un'area di memoria pre-allocata.

Il nome del vettore rappresenta *l'indirizzo iniziale* dell'area di memoria associata: `v ≡ & v[0]`.

Il nome del vettore ***non rappresenta l'intero vettore.***



Cosa succede allora se si passa un vettore come parametro a una funzione?

ESEMPIO

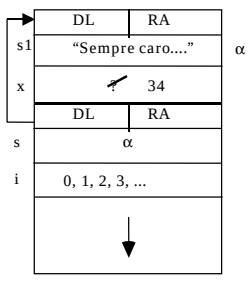
```
int lunghezza(char s[]);
main(){
  char s1[] = "Sempre caro mi fu quest'ermo colle";
  int x = lunghezza(s1);
}
```

Quando il main afferma di "passare `s1`" alla funzione `lunghezza`, *cosa le sta realmente passando?*

VETTORI PASSATI COME PARAMETRI (II)

Il nome del vettore rappresenta *l'indirizzo iniziale* dell'area di memoria associata.
↓
Quando si passa un vettore a una funzione **non si copia l'intero vettore, *si copia solo il suo indirizzo iniziale!***

```
int lunghezza(char s[]){
  int i;
  for (i=0; s[i]!='\0'; i++) ;
  return i;
}
main(){
  char s1[] = "Sempre caro mi fu quest'ermo colle";
  int x = lunghezza(s1);
}
```



VETTORI PASSATI COME PARAMETRI (III)

Quando si passa un vettore a una funzione **non si copia l'intero vettore, *si copia solo il suo indirizzo iniziale!***

↓

Agli occhi dell'utente, **i vettori sono passati *per indirizzo.***

MA ALLORA:

- se quello che viene passato, in realtà, è solo *l'indirizzo iniziale* del vettore,
- **nell'intestazione della funzione si può sostituire la notazione a vettore**

  <tipo> nomevettore [ ]
- **con la notazione a puntatore:**

  <tipo> * nomevettore

Questo **non cambia assolutamente nulla** in ciò che la macchina virtuale C fa: **rende solo più esplicito *ciò che accadrebbe comunque!***

ESEMPIO

```
int lunghezza(char * s){
  int i;
  for (i=0; s[i]!='\0'; i++) ;
  return i;
}
```


VETTORI PASSATI COME PARAMETRI (IV)

Se le notazioni

<tipo> nomevettore [ ] e **<tipo> * nomevettore**

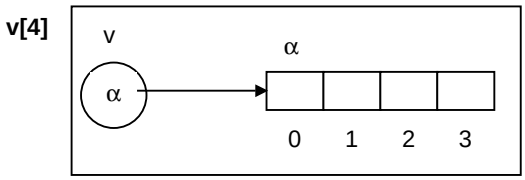
sono equivalenti,

cosa possiamo dire dei due operatori [ ] e * ?

- L'operatore `[ ]`, applicato a un *nome di vettore*, accede alla i-esima variabile del vettore.
- L'operatore `*`, applicato a un *puntatore*, accede alla variabile da esso puntata.

↓

Sono entrambi operatori di dereferenziamento



`*v ≡ v[0]`

e per estensione:

Aritmetica dei puntatori

`*(v+1) ≡ v[1]`

…

`*(v+i) ≡ v[i]`

L'operatore `[ ]` è sovrabbondante, esiste solo per comodità

→ gli operatori `*` e `[ ]` sono *intercambiabili !*

ARITMETICA DEI PUNTATORI

Le relazioni viste poco fa per i vettori:

`*v ≡ v[0]`

`*(v+i) ≡ v[i]`

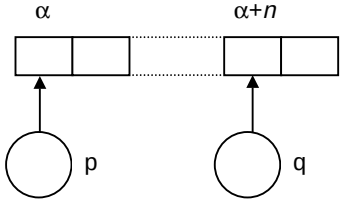
sono un'applicazione dell'***aritmetica dei puntatori.***

Più in generale:

se **`p`** è un puntatore a T, e **`n`** è un intero (eventualmente anche negativo),

l'espressione `p+n` denota un altro puntatore a T, che punta *"n celle dopo"* la cella puntata da `p`.

Se **`n`** è negativo, la cella denotata da **`p+n`** precede in realtà quella puntata da **`p`**)



Analogamente, se **`p`** e **`q`** sono due puntatori a T, l'espressione **`p-q`** denota il numero (intero) di celle che separano **`q`** da **`p`** (eventualmente negativo se in realtà **`q`** precede **`p`**).

NB: somme di puntatori, come **`p+q`**, sono illegali.

ESEMPIO 6

Scrivere *una funzione* che, dato un vettore di N interi, calcoli il valore massimo.

Si tratta di riprendere l'Esempio 1 ed esprimere come funzione ciò che prima era svolto direttamente nel main.

CODIFICA

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

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

NOTE:

- la funzione `findMax` potrebbe *modificare* il vettore `v`

  per evitarlo → **`const`** `int v[]`
- l'intestazione di `findMax` avrebbe anche potuto essere scrita come `int findMax(`**`const`**` int* v, int n)`
- quante variabili di nome `max` sono definite, e dove?

ESEMPIO 7

Scrivere una funzione che, data una stringa di caratteri, ne calcoli la lunghezza.

Si tratta di riprendere l'Esempio 3.

CODIFICA

```
int length(char s[]){
  int i;
  for (i=0; s[i]!='\0'; i++) ;
  return i;
}

main(){
  char s[] = "Nel mezzo del cammin di nostra vita";
  int l = length(s);
}
```

UNA CODIFICA ALTERNATIVA

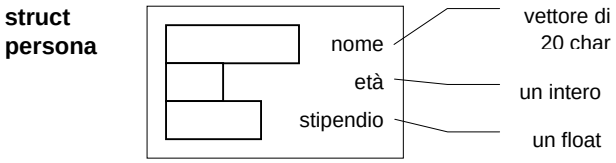
```
int length(char s[]){     /* o anche:  char *s   */
  char *s0 = s;
  while (*s!='\0') s++ ;
  return s-s0;       /* aritmetica dei puntatori */
}
```

Uno hacker compatterebbe il ciclo così:

```
  while (*s++) ;
```


STRUTTURE

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



SINTASSI di una *definizione* di una *variabile* di tipo struttura:

<tipo-struttura> <identificatore> ;

dove <tipo-struttura> è un costrutto della forma:

struct [<etichetta>] **{** ***{<definizione variabili>}*** **}**

ESEMPIO:

```
struct { int x, y; } p;
struct ptx {float x; int n; } p1, p2;
```

L' <etichetta> è opzionale: `ptx` *non* è un nome di variabile!

È anche possibile definire dei *tipi struttura:*

```
typedef struct { int x, y; } punto;
```

In questo modo, si può usare **punto** per dichiarare o definire variabili come se fosse un tipo predefinito:

```
punto px, py;
```


STRUTTURE (II)

Una volta definita una struttura,

come si accede alle variabili (campi) in essa contenute ?

NOTAZIONE PUNTATA

Se `p` è una variabile di un tipo struttura che definisce due campi di nome `x` e `y`, essi sono accessibili con la notazione:

p.x p.y

ESEMPIO

```
typedef struct { int x, y; } punto;
main(){
 punto p1 = {1,2}, p2 = {3,7}, p3;
 p3.x = p1.x + p2.x;
 p3.y = p1.y + p2.y;
}
```

UN ALTRO ESEMPIO

```
typedef struct { char nome[20]; int peso; } frutto;
main(){
 frutto f1 = {"mela",70}, f2 = {"arancio",50};
 int peso = f1.peso + f2.peso;  /* non è ambiguo */
}
```

NB: le strutture sono passate alle funzioni *per valore*, copiando byte per byte tutto il loro contenuto.

VETTORI PASSATI PER VALORE… ?

In pratica, dunque, in C un vettore è sempre passato per indirizzo, mai per valore.

Ma attenzione:

- non è che il C passi i vettori in modo diverso da tutto il resto *per scelta*
- è *il significato del nome del vettore* a provocare questo!

In particolare,

non è per efficienza che i vettori vengono passati per indirizzo

infatti, una struttura della stessa dimensione (o anche più grande...) viene *tranquillamente copiata!*

ESEMPIO

```
main(){
 struct {
  char s1[] = "Sempre caro mi fu quest'ermo colle";
 } miaStringa;

  int x = lunghezza(miaStringa);
}
```

MORALE: **per passare per valore un vettore è sufficiente "racchiuderlo" dentro a una struttura**

In effetti, così facendo *la struttura fornisce al vettore proprio ciò che gli manca*, cioè un "contenitore" dotato di nome che permetta di denotare il vettore *nella sua globalità.*