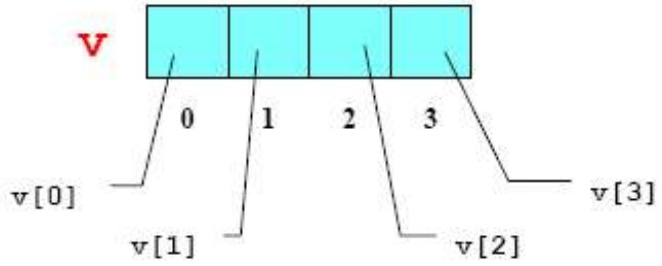
Tipi di dato STRUTTURATI

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

In C si possono *definire tipi strutturati: array e strutture*

[ ] *(array)*

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



Array (vettore)

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

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

Esempi:

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

ATTENZIONE: Sbagliato perchè Il compilatore non sa quanta memoria allocare per l'array

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

Esempio

lettura da input degli elementi di un vettore

```
#include <stdio.h>
#define N 3
main()
{ int k;
  int a[N];
  for(k=0; k < N; k++)
   {printf("Inserire valore %d: ",
  k);
    scanf("%d", &a[k]);
 }
```

define → definisce una costante di nome N e di valore 3

Esempio

Problema: **inizializzare un vettore con il prodotto degli indici posizionali dei suoi elementi**

```
#include <stdio.h>
#define N 3
main()
{ int i=0;
  int a[N];
  while (i<N){
   a[i]=i*i; /*gli elementi del vettore sono 0,1,4*/
   i++;
  }
}
```

Esempio: massimo

Scrivere un programma che, dato un vettore di N interi, ne 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 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 di tentativo***

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

Al termine, il valore del massimo di tentativo coincide col valore massimo ospitato nel vettore

$m_{n-1} \geq v[0], v[1] \ldots v[n-1]$

Esempio: massimo

Codifica:

```
#define DIM 4
main() {
    int v[] = {43,12,7,86};
    int i, max=v[0];
    for (i=1; i<DIM; i++)
       if (v[i]> max) max = v[i];
       /* ora max contiene il massimo */
}
```

Se vi è una *inizializzazione esplicita*, la dimensione dell'array ***può essere omessa***

Espressione di *inizializzazione* di un array

Dimensione Logica vs. Fisica

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*** alla sua ***dimensione fisica***

Spesso, la *porzione di array* realmente utilizzata *dipende dai dati d'ingresso*

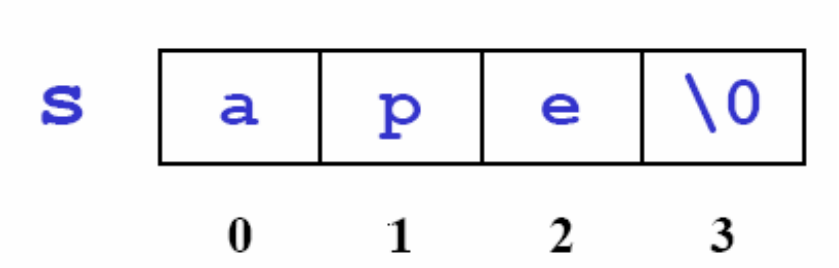
```
#define DIM 10
main() {
 int v[DIM] = {273,340,467,10};
```

Dimensione fisica = 10

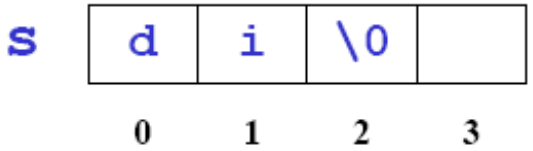
Dimensione logica = 4

Stringhe: array di caratteri

- Una ***stringa*** *di caratteri in C* è un ***array di caratteri terminato dal carattere '\0'***



- Un vettore di N caratteri può dunque ospitare stringhe *lunghe al più N-1 caratteri,* perché una cella è destinata al ***terminatore*** **'\0'**
- Un array di N caratteri può essere usato per memorizzare ***anche stringhe più corte di N-1***
- In questo caso, *le celle oltre la k-esima* (essendo K la lunghezza della stringa) *sono* ***logicamente vuote****:* sono inutilizzate e contengono un valore casuale



Stringhe: Inizializzazione

Una stringa si può *inizializzare,* come ogni altro array, elencando le singole componenti:

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

oppure anche, più brevemente, *con la forma compatta* seguente:

char s[4] = "ape" ; ⇨ Virgolette e non apici

Il carattere di terminazione **'\0'** è *automaticamente incluso* in fondo

Attenzione alla lunghezza: quando la dimensione delle celle finisce, i caratteri successivi non sono memorizzati

Stringhe: lettura e scrittura

Una stringa si può *leggere da tastiera e stampare,* come ogni altro array, elencando le singole componenti:

```
...
char str[4]; int i;
for (i=0; i < 3; i++)
    scanf("%c", &str[i]);
str[3] = '\0';
...
```

oppure anche, più brevemente, *con la forma compatta* seguente:

```
...
char str[4];
scanf("%s", str);
```

Esempio: confronto

Problema: Date due stringhe di caratteri, decidere quale precede l'altra in ordine alfabetico

Rappresentazione dell'informazione:

poiché vi possono essere *tre* risultati (s1<s2, s1==s2, s2<s1), *un boolean non basta*

Si può decidere di utilizzare ***un intero (negativo, zero, positivo)***

Specifica:

scandire uno a uno gli elementi ***di uguale posizione*** delle due stringhe, ***o fino alla fine delle stringhe, o fino a che se ne trovano due diversi***

- *nel primo caso, le stringhe sono uguali*
- *nel secondo, sono diverse*

nel secondo caso, confrontare i due caratteri così trovati, e determinare qual è il ***minore***

- la stringa a cui appartiene tale carattere ***precede l'altra***

Esempio: confronto

Codifica:

```
main() {
    char s1[] = ”Maria";
    char s2[] = ”Marta";
    int i=0, stato;

    while (s1[i]!='\0' && s2[i]!='\0' && s1[i]==s2[i])  i++;

    stato = s1[i]-s2[i];
    ……
}
```

negativo ↔ s1 precede s2
positivo ↔ s2 precede s1
zero ↔ s1 è uguale a s2

Esempio: copia

Problema: Data una stringa di caratteri, copiarla in un altro array di caratteri (di lunghezza non inferiore)

Ipotesi: La stringa è “ben formata”, ossia correttamente terminata dal carattere ‘\0’

Specifica:

- scandire la stringa, elemento per elemento, fino a trovare il terminatore ‘\0’ (che esiste certamente)
- *nel fare ciò, copiare l’elemento corrente nella posizione corrispondente dell’altro array*

Esempio: copia

Codifica: copia della stringa carattere per carattere

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

Al termine, occorre garantire che anche la nuova stringa sia “ben formata”, inserendo esplicitamente il terminatore

Esempio: copia

Perché non fare così?

```
main() {
   char s[] = "Nel mezzo del cammin di";
   char s2[40];
   s2 = s;
}
```

ERRORE DI COMPILAZIONE:
incompatible types in assignment

GLI ARRAY NON POSSONO ESSERE MANIPOLATI NELLA LORO INTEREZZA

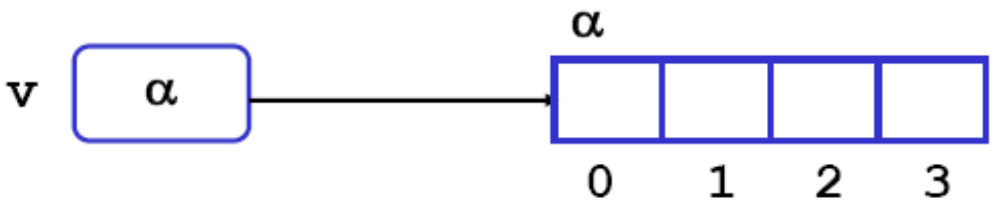
Array: implementazione

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

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 ≡ &v[0] ≡ α



Array: indirizzo del 1° elemento

- 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 è 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** ≡ **&v[0]** ≡ α)

Quindi, *passando un array a una funzione:*

- *non si passa l'intero array*
- si passa solo ***(per valore, come sempre)*** il suo indirizzo iniziale (**v** ≡ **&v[0]** ≡ α)

L'effetto finale apparente è che *l'array sia passato per riferimento*

Riassumendo …

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 appaiono trasferiti ***per riferimento***

Notazione a puntatore

Ma se quello che passa è solo *l'indirizzo iniziale* dell'array, che è un puntatore...

... allora ***si può adottare direttamente la notazione a puntatori nella intestazione della funzione***

In effetti, *l'una o l'altra notazione sono, a livello di linguaggio, assolutamente equivalenti*

- non cambia niente nel funzionamento
- si rende solo *più evidente ciò che accade comunque*

Esempio

Problema: Data una stringa di caratteri, *scrivere una funzione* che ne calcoli la lunghezza

Codifica:

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

Si può anche usare:
int lunghezza(char *s) {

Nel caso di notazione a puntatore,
s[lung] ≡ *(s+lung)

Nel caso delle stringhe, non è necessario indicare alla funzione **la dimensione dell'array** perché può essere dedotta dalla posizione dello '\0'

Operatori di deferenziazione

- L'operatore *, applicato a un *puntatore*, accede alla variabile da esso puntata
- L'operatore **[ ]**, applicato a un *nome di array e a un intero i*, accede alla i-esima variabile dell'array

Sono entrambi operatori di dereferencing

***v ≡ v[0]**

Aritmetica dei puntatori

Oltre a ***v** ≡ **v[0]**, vale anche:

***(v+1)** ≡ **v[1]**

…

***(v+i)** ≡ **v[i]**

Gli operatori * e [ ] *sono* *intercambiabili*

Espressioni della forma **p+i** vanno sotto il nome di ***aritmetica dei puntatori***, e denotano l'indirizzo posto ***i*** celle *dopo l'indirizzo denotato da* ***p*** ***(celle, non byte)***

Aritmetica dei puntatori (2)

Non solo sono consentite operazioni di somma fra puntatori e costanti intere ma anche:

Assegnamento e ***differenza*** fra puntatori

- **int *p,*q; p=q; p-q; p=p-q;**

La differenza però produce un warning

Altre operazioni aritmetiche fra puntatori non sono consentite:

- **int *p, *q; p=p*2; q=q+p;**

- Le operazioni sono ***corrette*** se i puntatori riferiscono lo ***STESSO TIPO*** (***non tipi compatibili***). Attenzione: comunque solo ***warning*** dal compilatore negli altri casi

Esempio: max

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()*

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

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 quando l'array termina (possibile indirizzamento scorretto). È possibile ometterlo, solo nel caso di stringhe.*

Esempio: max

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

Per assicurarsi che la funzione NON modifichi l'array (passato per riferimento), si può imporre la qualifica **const.** Se si tentano modifiche: *cannot modify a const object*

```
int findMax(const int v[], int dim) {
```

Libreria sulle strighe

- Il C fornisce una ricca libreria di funzioni per operare sulle stringhe:

```
#include <string.h>
```

Include funzioni per:

- copiare una stringa in un'altra (**strcpy**)
- concatenare due stringhe (**strcat**)
- confrontare due stringhe (**strcmp**)
- cercare un carattere in una stringa (**strchr**)
- cercare una stringa in un'altra (**strstr**)
- ...