AVEVAMO LASCIATO IN SOSPESO …

Perché non fare così?

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

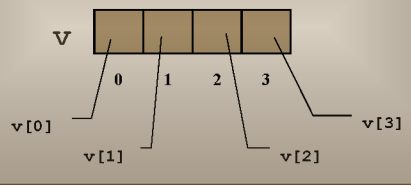
ERRORE DI COMPILAZIONE:
left operand must be l-value

PERCHÉ GLI ARRAY NON POSSONO ESSERE MANIPOLATI NELLA LORO INTEREZZA !

Vedremo più avanti il perché

ARRAY: STRUTTURA FISICA

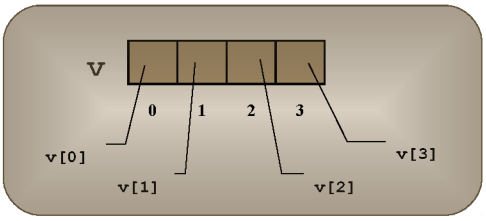
Un *array* è una *collezione finita di N variabili dello stesso tipo*, ognuna identificata da un *indice* compreso fra 0 e N-1



Praticamente, le cose non stanno proprio così.

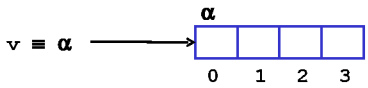
Se fosse così …

- Se fosse così, il nome dell'array rappresenterebbe tutto l'array:
- v ≡



ARRAY: STRUTTURA FISICA

- In C il nome di 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`

Assegnamento fra array?

```
main()
{ char s[] = "ciao";
  char s2[5];
    s2 = s;
}
```

left operand must be l-value

s[0]	'c'	1000
s[1]	'i'	1001
s[2]	'a'	1002
s[3]	'o'	1003
s[4]	'\0'	1004
s2[0]		1005
s2[1]		1006
s2[2]		1007
s2[3]		1008
s2[4]		1009

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:

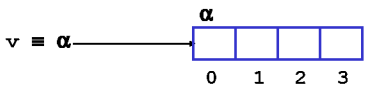
  È impossibile denotare un array nella sua globalità, in qualunque contesto.

- Quindi non 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`)



OPERATORE [ ]

```
char v[5]="ciao";
printf("%c",v[2]);
```

1000	'c'
1001	'i'
1002	'a'
1003	'o'
1004	'\0'

In pratica il C, per valutare `v[x]`, esegue questi passi:

1. Considera l'indirizzo `v` — 1. `v` = 1000
2. Considera la dimensione *d* in byte del tipo considerato — 2. `char` → *d* = 1 byte
3. Avanza, rispetto a `v`, di `x`**d* byte — 3. `v[2]` è la cella con indirizzo 1000+2*1=1002

OPERATORI DI DEREFERENCING

- 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]`

Array come parametri: sintassi

- Si può passare un array come parametro, usando la normale sintassi C

```
int somma(int a[4])
{ int i,s=0;
  for (i=3;i>=0;i--)
      s = s + a[i];
  return s;
}
main()
{int vet[4] = {1,3,2,5};
int s;
s = somma(vet);
}
```

Nella definizione si definisce il parametro come variabile di tipo array

Nell'invocazione si deve passare una variabile di tipo array

Ricordiamo la sintassi

- `int a[4];`
  vuol dire *"*`a` *è un array di 4 interi"*
- `printf("%d",a[4]);`
  vuol dire passa alla `printf` *"l'elemento di indice 4 dell'array* `a`*"*
- Se voglio passare l'intero array non devo mettere le quadre

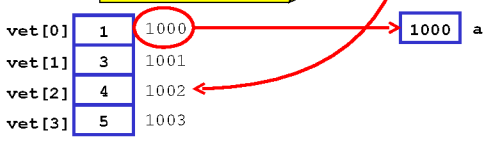
```
char s[5]="ciao";
printf("%s",s);
```

Array come parametri

```
main()
{int vet[4]={1,3,4,5};
 int s;
 s = fun( 1000 );
}
```

```
int fun(int a[4])
{ return *(1000+2) ;
}
```

Il nome dell'array rappresenta l'indirizzo iniziale, quindi sto comunicando al servitore un indirizzo!



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

  (`vet` ≡ `&vet[0]` ≡ α)

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

ESERCIZIO

- Si legga da tastiera una stringa che contiene al più 5 caratteri (incluso il terminatore)
- Si scriva una funzione che calcola la lunghezza della stringa inserita
- Si visualizzi il risultato
- Si mostri come viene eseguito il programma con i record di attivazione

ESEMPIO

Problema:

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

Codifica:

La dimensione non serve, perché tanto viene passato solo l'indirizzo iniziale (non tutto l'array)

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

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.*

NOTAZIONE A PUNTATORI

- 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

Da così...

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

... a così:

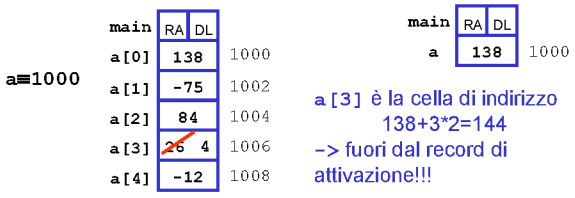
```
int lunghezza(char *s)
{ int lung=0;
  for (lung=0; s[lung]!='\0'; lung++);
  return lung;
}
```

Differenza fra array e puntatori

- Posso fare così?

```
main()
{ short int a[5];
  a[3]=4;
}
```

```
main()
{ short int *a;
  a[3]=4;
}
```

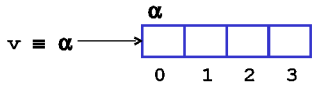


a[3] è la cella di indirizzo 138+3*2=144
-> fuori dal record di attivazione!!!

Differenza fra array e puntatori

- Dichiarare un array significa allocare un'area di memoria sufficiente a contenere tutte le celle dell'array

```
int v[4];
```



- Dichiarare un puntatore significa allocare un'area di memoria sufficiente a contenere un indirizzo

```
int *v;
```

ESEMPIO

Scrivere un programma che

- legge da tastiera un intero *n* e un array di *n* interi, tramite una funzione
- calcola l'indice del massimo dell'array tramite una funzione
- visualizza sullo schermo l'indice del massimo

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:

```
main()
{int max, n, v[4];
 n = lettura(v);
 max = findMax(v, n);
}
```

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 !*

ESERCIZIO

Scrivere un programma che

- legge da tastiera un array di interi, tramite una procedura
- legge da tastiera un intero *x*
- cerca l'elemento *x* nell'array tramite una funzione che fornisce
  - la posizione dell'elemento,
  - oppure -1 se l'elemento non è presente nell'array
- visualizza sullo schermo la posizione dell'elemento *x* se è presente (o "non esiste" se non è presente)

Funzione ricerca

```
int ricerca(int A[],int x, int n)
{ int i=0,trovato=0;
  while ((i<n) && (!trovato))
     if (A[i]==x)
          trovato=1;
     else i++;
  if (trovato)
     return i;
  else return -1;
}
```

E i vantaggi del passaggio per valore?

- Quindi se mi sbaglio e modifico il valore di un parametro di tipo array, la modifica avviene anche dal lato del chiamante

```
int ricerca(int A[],int x, int n)
{ int i=0,trovato=0;
  while ((i<n) && (!trovato))
     if (A[i] = x)
        trovato=1;
     else i++;
  if (trovato)
       return i;
  else return -1;
}
```

QUALIFICATORE CONST

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 ricerca(const int A[],int x, int n)
{ int i=0,trovato=0;
  while ((i<n) && (!trovato))
     if (A[i] = x)
        trovato=1;
     else i++;
  if (trovato)
       return i;
  else return -1;
}
```

cannot modify a const object

ESERCIZIO

- Dato un array di interi, si scriva una funzione che verifica se ci sono elementi ripetuti.
  - Se ci sono elementi ripetuti restituisce 1
  - Se non ci sono restituisce 0

ESERCIZIO

- Dato un array di interi, si scriva una procedura che elimina gli eventuali elementi ripetuti.

ESEMPIO

Problema:
Scrivere *una procedura* che copi una stringa in un'altra.

Codifica:

```
void strcpy(char dest[], char source[])
{ int i;
  for (i=0; source[i]!=0; i++)
  dest[i]=source[i];
  dest[i]=0;
}
```

ESERCIZIO: MERGE

- Si leggano da tastiera due array di interi ordinati
- Si scriva una funzione `merge` che crea il vettore ordinato che contiene gli elementi di entrambi gli array

STRUTTURE

- A differenza di quanto accade con gli array, il nome della struttura rappresenta la struttura nel suo complesso.

Quindi, è possibile:

- assegnare una struttura a un'altra (`f2 = f1`)
- che una funzione restituisca una struttura

E soprattutto:

- passare una struttura come parametro a una funzione significa passare una copia

ASSEGNAMENTO TRA STRUTTURE

```
typedef struct
{char nome[20]; int  peso;
} frutto;

main()
{ frutto f1, f2;
  strcpy(f2.nome,"mela");
  f2.peso=70;
  f1 = f2;
}
```

Equivale a copiare `f2.peso` in `f1.peso`, e `f2.nome` in `f1.nome`.

STRUTTURE PASSATE COME PARAMETRI

- Il nome della struttura rappresenta, come è naturale, *la struttura nel suo complesso*
- quindi, non ci sono problemi nel passarle come parametro ad una funzione: *avviene il classico passaggio per valore*
  - tutti i campi vengono copiati, uno per uno!
- è perciò possibile anche *restituire come risultato* una struttura

ESEMPIO

Tipo del valore di ritorno della funzione.

```
frutto macedonia(frutto f1, frutto f2)
{     frutto f;
      f.peso = f1.peso + f2.peso;
      strcpy(f.nome, "macedonia");
      return f;
}
```

La funzione di libreria `strcpy()` copia la costante stringa `"macedonia"` in `f.nome`.

Trucco …

- Ma allora, se voglio fare
  - il passaggio per copia di un array
  - oppure restituire un array come risultato di una funzione
  - oppure fare l'assegnamento fra array

  posso includere l'array in una struttura!

```
typedef struct
{ int a[5];
} array;
main()
{ array x,y;
  x.a[1]=0; x.a[2]=3;
  y=x;
}
```

Esercizio (28 mar 2007)

Si scriva una funzione (ricorsiva) C con la seguente interfaccia

```
int nullo(int a[], int n);
```

che fornisce come valore di ritorno

- 1 se tutti i valori contenuti nell'array `a` con indice fra 0 ed n sono uguali a zero,
- 0 altrimenti (cioè se almeno uno degli elementi dell'array `a` con indice compreso fra 0 e `n` è diverso da zero).

(continua)

Si mostri poi il funzionamento del seguente programma, che invoca la funzione definita precedentemente, utilizzando i record di attivazione.

```
int q( int a[], int *t , int h)
{ int s=1;
  if (!nullo(a,*t))
      for (h=0; a[h]==0; a[h]++)
          h++;
  (*t) = h;
  return s + h ;
}
main ()
{ int a[3]={0 ,7 ,0}, t=2, n=2;
  a[n]= q(a,&t,n);
}
```

ESERCIZIO

- Uno studente deve calcolare la media dei suoi esami per la laurea. Gli esami non sono tutti uguali: a ciascun esame è associato un numero di crediti. La media degli esami è la media pesata, considerando i crediti come pesi.
- Es
  - Fondamenti Informatica 1 6 crediti, voto 30
  - Inglese 3 crediti, voto 20
  - media = (30*6 + 20*3) / (6+3) = 26.67
- Si scriva un programma che
  - definisce un tipo di dato "esame" che contiene sia il numero di crediti sia il voto dell'esame
  - legge da tastiera i dati sugli esami sostenuti da uno studente (crediti e voto per ogni esame)
  - calcola la media pesata