



# Tipi di Dato e Array

Informatica B AA 2017 / 2018

Giacomo Boracchi

11 Ottobre 2017

[giacomo.boracchi@polimi.it](mailto:giacomo.boracchi@polimi.it)



## Warm up

Scrivere un programma per conteggiare quanto la vostra aula ha speso in totale per il pranzo Venerdì scorso.

Calcolare la spesa media per il pranzo



## Altri quesiti

Altre domande:

- chi ha speso di più di tutti
- se qualcuno ha speso più di tutti gli altri messi assieme
- Si supponga di «fare alla romana» e che quindi tutti devono pagare il prezzo medio. Dire a chi ha pagato di quanto deve ricevere e a chi ha pagato quanto deve versare.

Per rispondere all'ultima domanda servirebbe **tener traccia** di quanto viene «versato» da ciascuno (i.e. i valori assegnati alla variabile `soldi`).

Riprendendo il paragone variabili-foglietti su cui scrivere, servirebbe, al posto di un foglietto `soldi`, una **sequenza di foglietti**, ed ciascun foglietto tiene traccia dei valori inseriti

**Quindi sequenze di variabili: gli array**



# Tipi di dato



I **tipi di dato** rappresentano:

- un insieme di **valori**
- un insieme di **operazioni** applicabili a questi

Ogni **tipi di dato diversi** hanno **rappresentazioni** in memoria differenti

- Il numero di celle/parole e la codifica utilizzata può cambiare

La memoria utilizzata per allocare le variabili di un determinato tipo cambia con la piattaforma (i.e., compilatore / sistema operativo / hardware)



## Tipi di dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile** (e.g. **int**, **char**, **float**..)
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
- Altra classificazione:
  - **Built in**, tipi già presenti nel linguaggio base
  - **User defined**, nuovi tipi creati nei programmi «componendo» variabili di tipo built in



## Tipi di dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile** (e.g. **int, char, float..**)
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
- Altra classificazione:
  - **Built in**, tipi già presenti nel linguaggio base
  - **User defined**, nuovi tipi creati nei programmi «componendo» variabili di tipo built in



## Variabili e Tipo di Dato

- In C **tutte le variabili** hanno un **tipo**, associato stabilmente mediante la **dichiarazione**
- Il **tipo** di una variabile:
  - definisce l'insieme dei **valori ammissibili**
  - definisce l'insieme delle **operazioni applicabili**
  - permette di **rilevare errori** al momento della compilazione
  - definisce lo **spazio in memoria** allocato in corrispondenza alla variabile
    - Questa però dipende anche dalla piattaforma (i.e., compilatore + sistema operativo + hardware)





## Tipi Semplici

- `char`, `int`, `float`, `double`



## Tipi Semplici

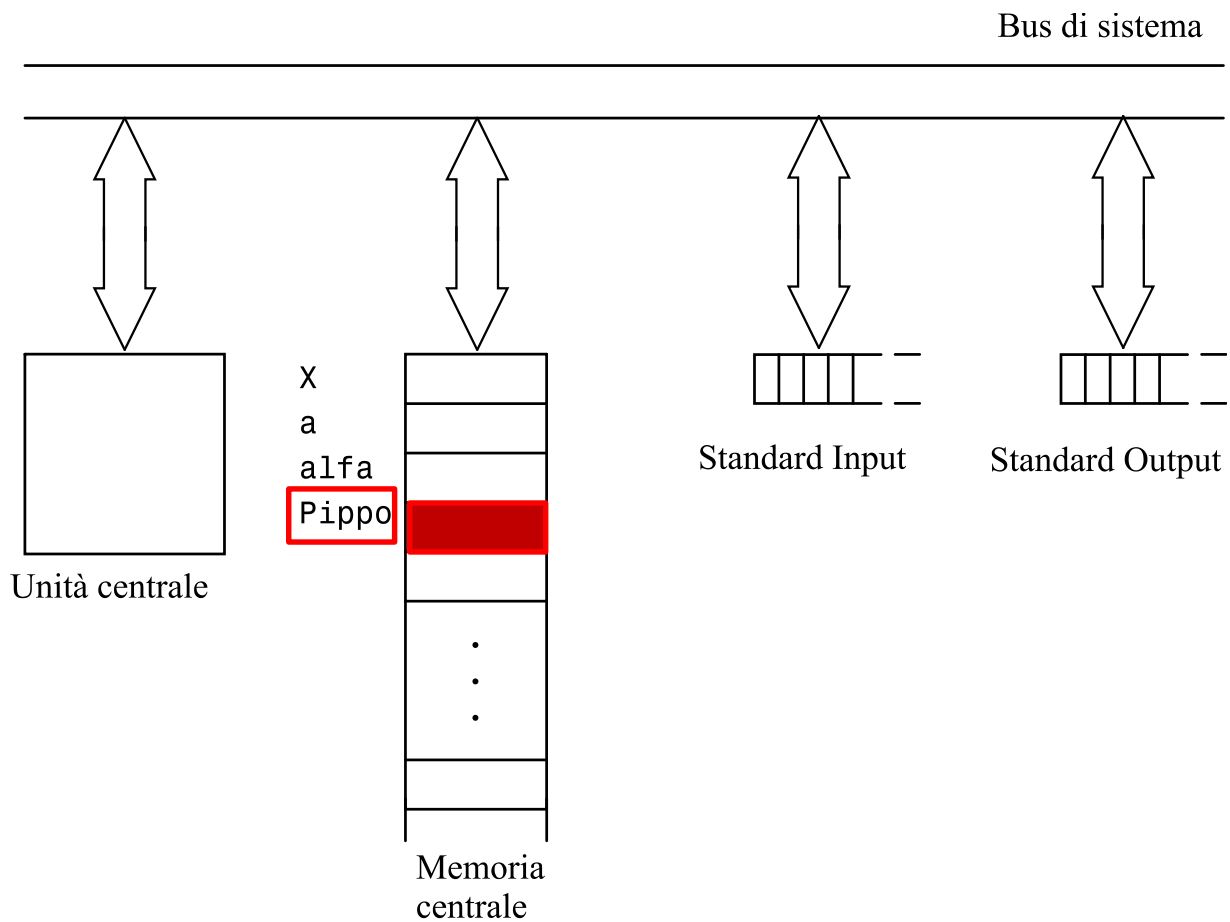
- Ecco i **quattro tipi semplici** del C e la loro dimensione
  - **char**: 1 Byte
  - **int**: tipicamente 1 parola di memoria
  - **float**: dipende dal compilatore (4 Byte spesso)
  - **double**: dipende dal compilatore (più del **float** )
- **Qualificatori di tipo** (per **int** e **char**)
  - **signed** utilizza una codifica con il segno
  - **unsigned** prevede solo valori positivi

NB Allocano lo stesso spazio
- **Quantificatori di tipo**, modificano la dimensione allocata
  - **short** (per **int**)
  - **long** (per **int** e **double**)



# I Qualificatori, Quantificatori e lo spazio allocato

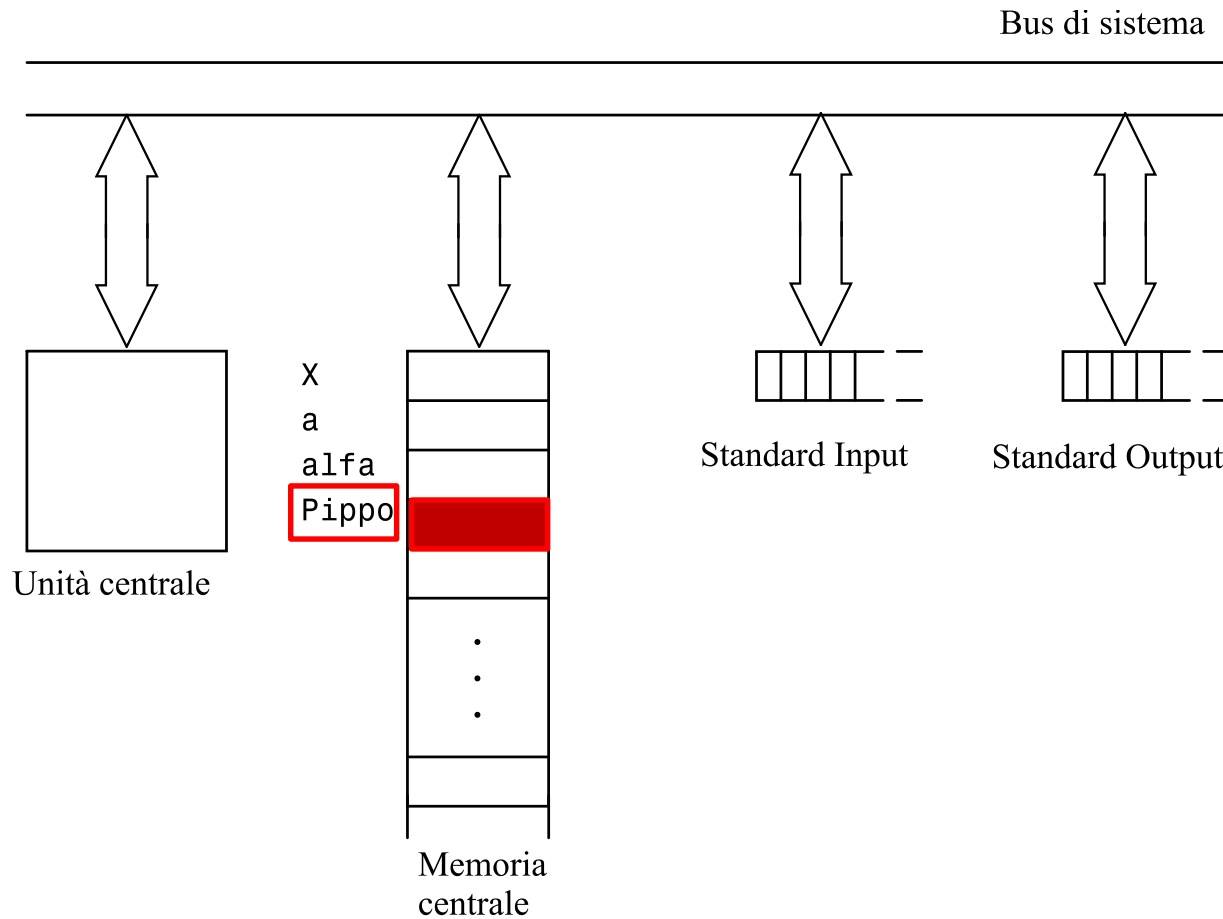
```
int Pippo;
```





# I Qualificatori, Quantificatori e lo spazio allocato

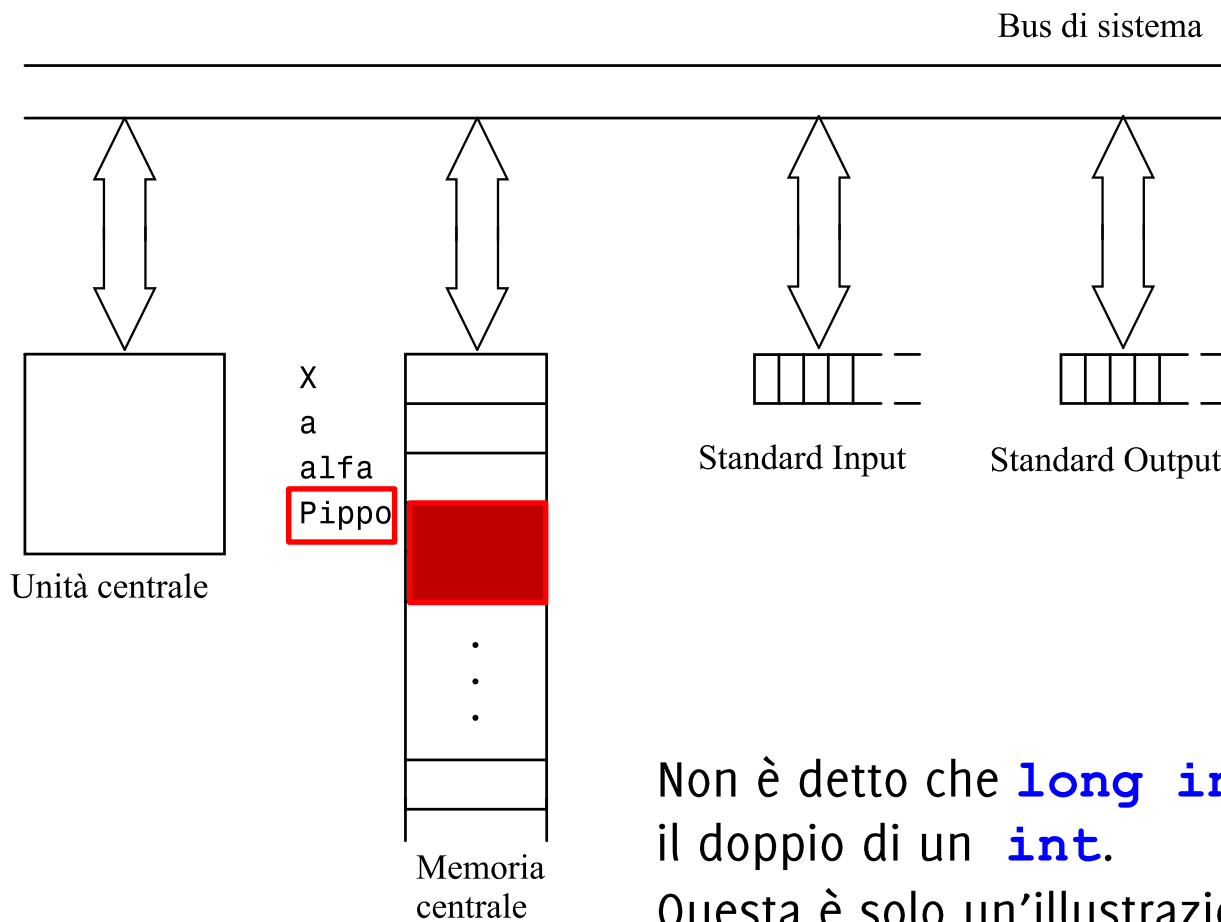
`unsigned int Pippo;`





# I Qualificatori, Quantificatori e lo spazio allocato

```
long int Pippo;
```

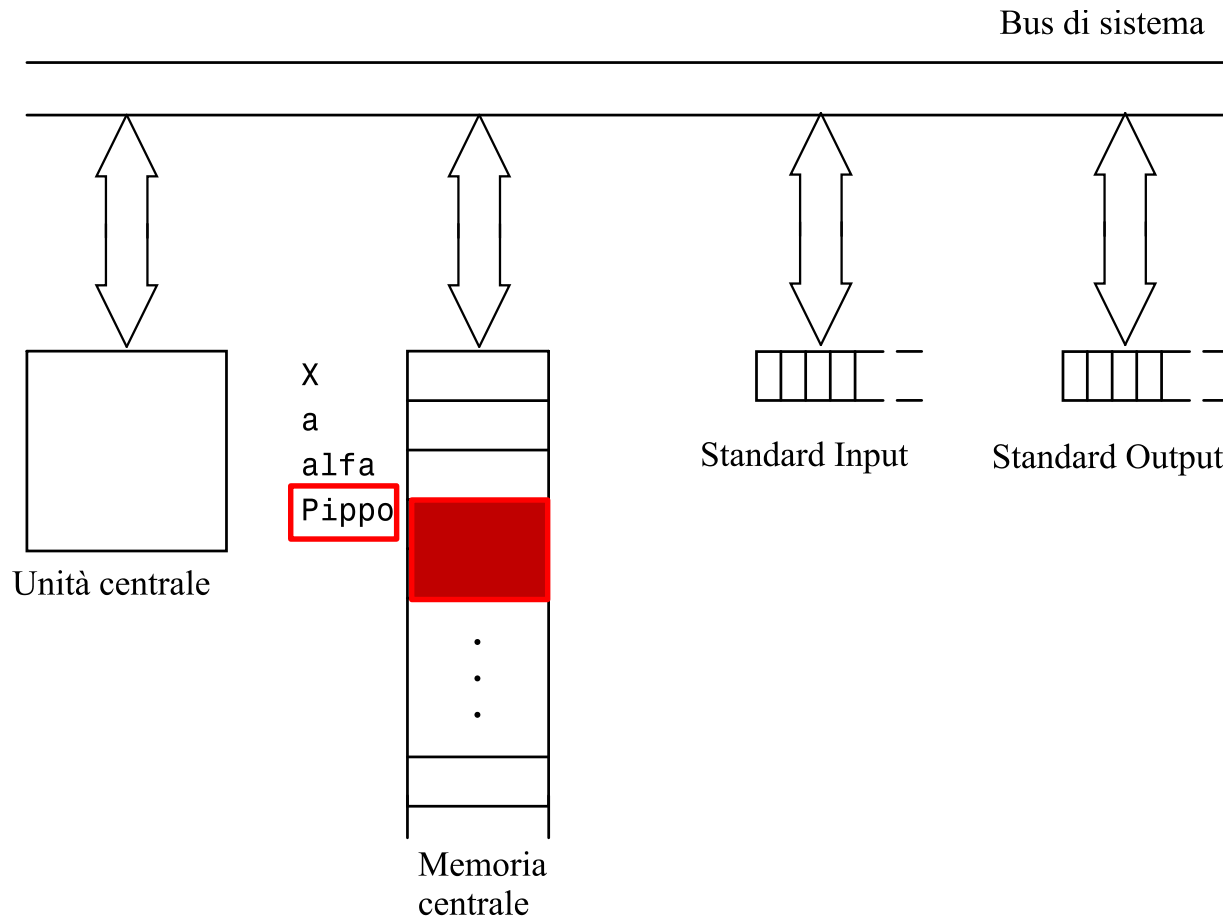


Non è detto che `long int` richieda il doppio di un `int`.  
Questa è solo un'illustrazione



# I Qualificatori, Quantificatori e lo spazio allocato

```
unsigned long int Pippo;
```





## Il tipo `int`

- Rappresentano un sottoinsieme di  $\mathbb{N}$
- Lo **spazio allocato** è tipicamente **una parola**, e dipende dalla piattaforma, oltre che dai qualificatori e quantificatori
- Fatti garantiti:
  - spazio (**short int**)  $\leq$  spazio (**int**)  $\leq$  spazio (**long int**)
  - spazio (**signed int**) = spazio (**unsigned int**)
- Es, se la parola è a 32 bit,
  - **signed int**  $\{-2^{31}, \dots, 0, \dots, +2^{31} - 1\}$ , i.e.,  $2^{32}$  numeri
  - **unsigned int**  $\{0, \dots, +2^{32} - 1\}$ , sempre  $2^{32}$  numeri
- Come faccio a sapere i limiti per un intero?
  - **#include<limits.h>**, e richiamo le costanti **INT\_MIN**, **INT\_MAX**
- Quando il valore di una variabile **int** eccede **INT\_MAX** si ha **overflow**



## Operazioni built-in per dati di tipo `int`

- `=` Assegnamento di un valore `int` a una variabile `int`
- `+` Somma (tra `int` ha come risultato un `int`)
- `-` Sottrazione (tra `int` ha come risultato un `int`)
- `*` Moltiplicazione (tra `int` ha come risultato un `int`)
- `/` Divisione con troncamento della parte non intera (risultato `int`)
- `%` Resto della divisione intera
- `==` Relazione di uguaglianza
- `!=` Relazione di diversità
- `<` Relazione “minore di”
- `>` Relazione “maggiore di”
- `<=` Relazione “minore o uguale a”
- `>=` Relazione “maggiore o uguale a”

Operatori  
Aritmetici

Operatori  
Relazionali





## Il tipo `float` e `double`

- Approssimazione di  $\mathbb{R}$  (che è un insieme denso), quindi i **valori** vengono approssimati per «magnitudine», e limiti nella **precisione** della rappresentazione
- Nella rappresentazione in virgola mobile (floating point) il numero  $n$  si scrive come due parti separate da “E”:
  - $m$ : mantissa
  - $e$ : esponente (rispetto alla base 10), tali che  $n = m * 10^e$

Ad esempio, 1 780 000.000 0023 in virgola mobile:

178 000.000 000 23 E1

17 800 000 000 023 E-7

1.780 000 000 0023 E+6

ecc.



## Spazio su **float** e **double**

Unico fatto certo:

spazio (**float**)  $\leq$  spazio (**double**)  $\leq$  spazio(**long double**)

Esempio tipico **float** in 4 byte e **double** in 8 byte

$\Rightarrow$  accuratezza:           6 decimali per **float**

15 decimali per **double**

$\Rightarrow$  valori                   tra  $10^{-38}$  e  $10^{+38}$  per **float**

tra  $10^{-308}$  e  $10^{+308}$  per **double**



## Operazioni built-in per dati di tipo `float`

- `=` Assegnamento di un valore `float` a una variabile `float`
- `+` Somma (tra `float` , risultato `float`)
- `-` Sottrazione (tra `float` , risultato `float`)
- `*` Moltiplicazione (tra `float`, risultato `float`)
- `/` Divisione (tra `float`, risultato `float`)
- `==` Relazione di uguaglianza
- `!=` Relazione di diversità
- `<` Relazione “minore di”
- `>` Relazione “maggiore di”
- `<=` Relazione “minore o uguale a”
- `>=` Relazione “maggiore o uguale a”

Operatori  
Aritmetici

Operatori  
Relazionali



## Operazioni tra `float`

- operazioni applicabili a `float` (anche a `double` e `long double`) sono le stesse degli `int`, ma divisione `'/'` dà risultato reale
- **NB:** il simbolo dell'operazione è identico a divisione intera
- standard library `math.h` fornisce funzioni predefinite (`sqrt`, `pow`, `exp`, `sin`, `cos`, `tan...`) applicate a valori double  
⇒ si usa `double` anche quando basta `float`



## Il tipo `float` e `double` : le approssimazioni

- Nella rappresentazione di un numero decimale possono esserci **errori di approssimazione**
  - Non sempre:  $(x / y) * y == x$
  - Per verificare l'uguaglianza tra `float` o `double`, definire dei bounds : Invece di
    - `if (x == y) ...` è meglio
    - `if (x <= y + .000001 && x >= y - .000001)`
- Buona parte delle operazioni algebriche eseguibili tra `float` (es. l'elevamento a potenza, il logaritmo, la radice, il valore assoluto... ) sono nella libreria `math` che occorre includere con
  - `#include<math.h>`



## Il tipo `char`

- La codifica ASCII prevede di allocare sempre 1 Byte per rappresentare caratteri
  - alfanumerici
  - di controllo (istruzioni legate alla visualizzazione),
- C'è una corrispondenza tra i `char` e 256 numeri interi
- Le operazioni sui `char` sono le stesse definite su `int`
  - hanno senso gli operatori aritmetici (+ - \* / %)
  - hanno senso gli operatori di relazione (== , > , < ,... etc)
- `unsigned char` coprono l'intervallo [0, 255].
- `signed char` coprono l'intervallo [-128, 127].
- **N.B.** non esistono tipi semplici più «piccoli» del `char`



## Il tipo `char`

- I valori costanti di tipo `char` nel codice sorgente si delimitano tra apici singoli `' '`
- Gli apici doppi `" "` vengono utilizzati per delimitare stringhe, i.e. sequenze di caratteri (non hanno un loro tipo built in)
  - le abbiamo già viste in `printf` e `scanf`



## La codifica ASCII (parziale)

DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR
48	0	65	A	75	K	97	a	107	k
49	1	66	B	76	L	98	b	108	l
50	2	67	C	77	M	99	c	109	m
51	3	68	D	78	N	100	d	110	n
52	4	69	E	79	O	101	e	111	o
53	5	70	F	80	P	102	f	112	p
54	6	71	G	81	Q	103	g	113	q
55	7	72	H	82	R	104	h	114	r
56	8	73	I	83	S	105	i	115	s
57	9	74	J	84	T	106	j	116	t
				85	U			117	u
				86	V			118	v
				87	W			119	w
				88	X			120	x
				89	Y			121	y
				90	Z			122	z





## Il tipo `char` esempi

```
char a,b;
```

```
b = 'q';
```

```
a = "q";
```

```
a = '\n';
```

```
b = 'ps';
```

```
a = 75;
```

```
a = 'c' + 1;
```

```
a = 'c' - 1;
```

```
    a = 20;
```

```
    a *= 4;
```

```
    a -= 10;
```

```
a = '1';
```



## Il tipo `char` esempi

```
char a,b;
```

```
b = 'q'; /* Le costanti di tipo carattere si  
         indicano con ' */
```

```
✘ a = "q"; /* NO: "q" è una stringa, anche se di  
          un solo carattere */
```

```
a = '\n'; /* OK: \n è un carattere a tutti gli  
          effetti anche sono due elementi*/
```

```
✘ b = 'ps'; /* NO: 'ps' non è un carattere valido*/
```

```
a = 75; /*associa ad a il carattere 'K' cfr ASCII
```

```
a = 'c' + 1; /* a diventa 'd' */
```

```
a = 'c' - 1; /* a diventa 'b' */
```

```
a = 20;
```

```
a *= 4; /* sta per a = a * 4, quindi a = 80 ('P')*/
```

```
a -= 10; //a <--70 che corrisponde al carattere 'F'
```

```
a = '1'; /*a è il carattere 1, corrispondente a 49
```



## Riepilogando sui tipi built in

- I tipi **integral** sono discreti, rappresentano valori **numerabili**
  - sono **char** ed **int** con tutti i qualificatori e quantificatori (**signed/unsigned char**, **short/long int**, **signed/unsigned int**)
- I tipi **floating** approssimano insiemi **densi**
  - sono **float** e **double**, eventualmente con il quantificatori **long**
- I tipi **floating** e **integral** assieme compongono i tipi **arithmetic**



# Tipi di Dato Strutturati

- Gli array



## I Tipi Strutturati in C

- Permettono di immagazzinare informazione aggregata
  - Vettori e matrici in matematica
  - Testi (sequenza di caratteri)
  - Immagini
  - Rubriche
  - Archivi,.. etc.
- Le variabili strutturate memorizzano diversi elementi informativi:
  - omogenei
  - eterogenei
- Oggi vedremo gli **array**



## Il Costruttore Array

Gli array sono **sequenze di variabili omogenee**

- **sequenza:** hanno un ordinamento (sono indicizzabili)
- **omogenee:** tutte le variabili della sequenza sono dello stesso tipo

Ogni elemento della sequenza è individuato da un indice



## Il Costruttore Array

Sintassi dichiarazione di una variabile mediante costruttore array

```
tipo nomeArray[Dimensione] ;
```

- **tipo** la keyword di un tipo (built in o user-defined)
- **nomeArray** è il nome della variabile
- **Dimensione** è un numero che stabilisce il numero di elementi della sequenza.

**NB: Dimensione** è un numero fisso, noto a compile-time:

- non può essere una variabile (il suo valore sarebbe definito solo a run-time)
- non è possibile modificare le dimensioni durante l'esecuzione (e.g. allungare o accorciare l'array)



## Il Costruttore Array, esempi

### Esempi

- `int vet[8];`
- `char stringa[5];`
- `float resti[8];`

**vet**

134
34
123
43215
2365
-145
523
45

**stringa**

'a'
'K'
'\n'
'3'
'\t'

**resti**

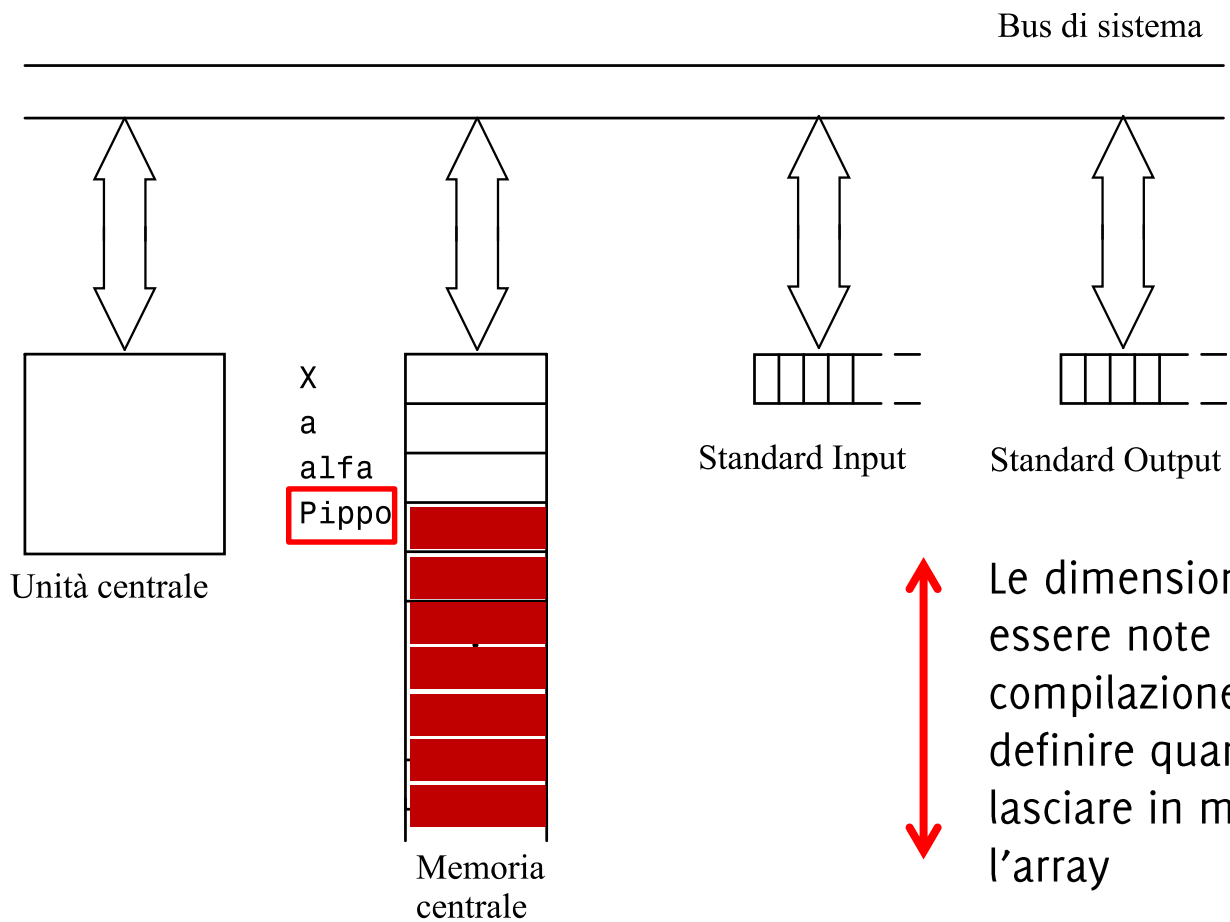
2.45
3.24
4.23
1245.2
236.5
-5.0
43.53
0





# Lo spazio allocato per gli array

- `int Pippo[20];`



Le dimensioni devono essere note prima della compilazione per definire quanto spazio lasciare in memoria per l'array



## Accedere agli elementi dell'array

- È possibile accedere agli elementi dell'array specificandone un **indice** tra parentesi quadre `[]`

```
int vet[20];
```

`vet[0]` è il primo elemento della sequenza

`vet[19]` è l'ultimo elemento della sequenza

- Ogni elemento dell'array è una **variabile** del **tipo** dell'array:  
`vet[7]` conterrà un valore intero
- Una volta **fissato l'indice**, non c'è differenza tra un elemento dell'array ed una qualsiasi **variabile** dello stesso tipo

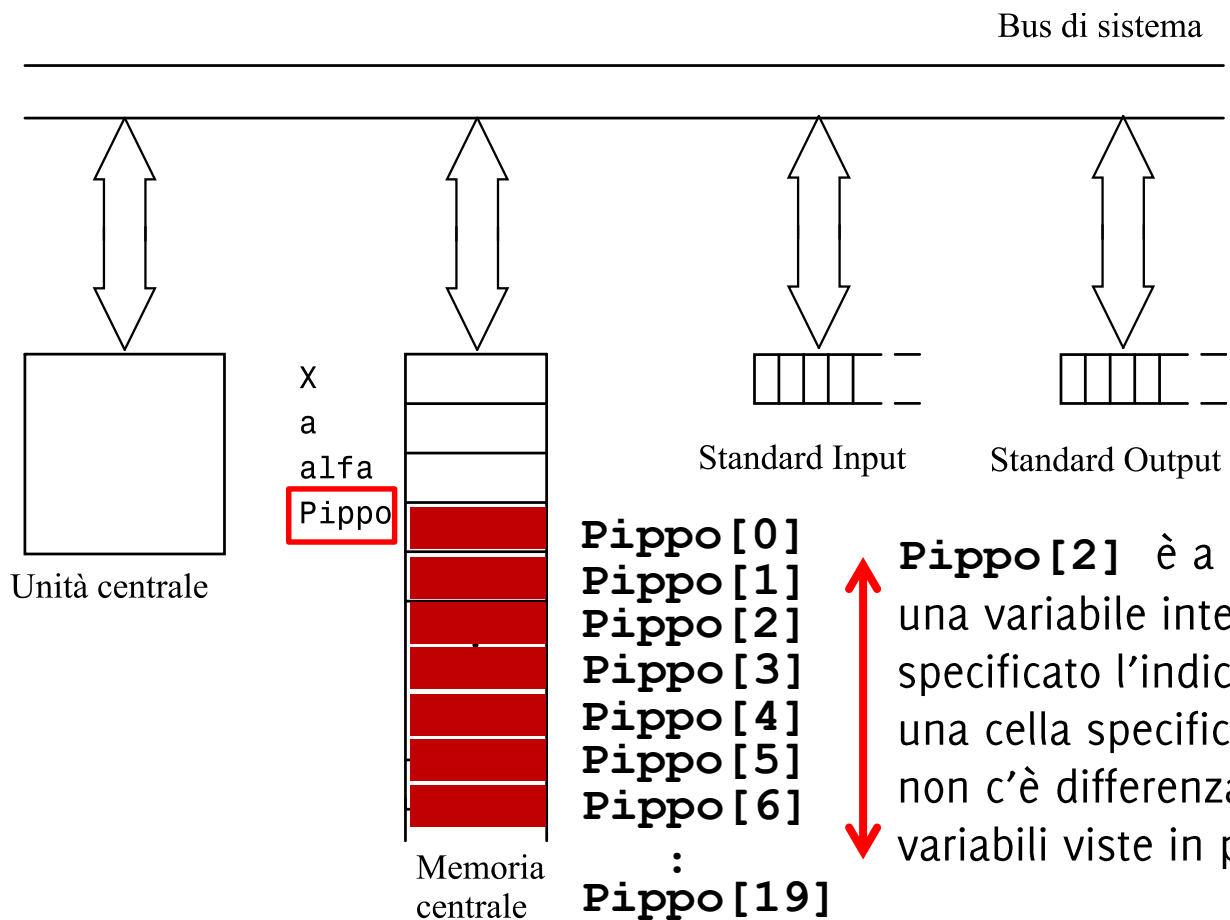
```
int a; a = vet[0]; vet[0] = a; vet[0] += a;
```

**NB** in C gli array sono indicizzati a partire da 0



# Lo spazio allocato per gli array

```
int Pippo [20] ;
```



**Pippo [2]** è a tutti gli effetti una variabile intera. Una volta specificato l'indice accedo ad una cella specifica dell'array e non c'è differenza con le variabili viste in precedenza



## Accedere agli elementi dell'array

Il valore dell'indice è di tipo integral (**char** , **int**)

È quindi possibile utilizzare una **variabile** per definire l'indice all'interno dell'array

```
int vet[20]; int i = 0;
```

L'espressione: **vet[i]**

va interpretata nel seguente modo:

1. Leggi il valore di **i**
2. Accedi all'elemento di **vet** alla posizione di indice **i**
3. Leggi il valore che trovi in quella cella di memoria (**vet[i]**)

con lo stesso criterio posso interpretare

```
vet[i + 1];
```



## Esempi di Operazioni su Array

Una volta fissato l'indice in un array si ha una variabile del tipo dell'array che può essere usata per

- assegnamenti

```
vet[2] = 7; vet[4] = 8 % 3;
```

```
i = 0; vet[i] = vet[i+1];
```

- operazioni logiche

```
vet[0] == vet[9]; vet[1] < vet[4];
```

- operazioni aritmetiche

```
vet[0] == vet[9] / vet[2] + vet[1] / 6;
```

- operazioni di I/O

```
scanf("%d", &vet[9]);
```

```
printf("valore pos %d = %d", i, vet[i]);
```



## Assegnamento tra Array

*Es:* Scrivere un frammento di codice per dichiarare un array di dimensione 3 e per scrivere in ogni variabile un numero (da 1 a 3) corrispondente alla posizione della cella.



## Assegnamento tra Array

*Es:* Scrivere un frammento di codice per dichiarare un array di dimensione 3 e per scrivere in ogni variabile un numero (da 1 a 3) corrispondente alla posizione della cella.

```
int vet[3];
```

```
vet[0] = 1;
```

```
vet[1] = 2;
```

```
vet[2] = 3;
```



## .. e senza Array

```
int a,b,c;           int vet[3];  
a = 1;              vet[0] = 1;  
b = 2;              vet[1] = 2;  
c = 3;              vet[2] = 3;
```

vs

Come faccio a richiamare "il secondo valore inserito"?

- Con le variabili devo salvare da qualche parte che **a** contiene il primo valore, **b** il secondo... perché le variabili non hanno un ordinamento
- Con il vettore mi basta accedere a **vet[1]** perché gli elementi di un vettore seguono un ordinamento





## .. e senza Array

```
int a,b,c;           int vet[3];  
a = 1;              vet[0] = 1;  
b = 2;              vet[1] = 2;  
c = 3;              vet[2] = 3;
```

vs

La soluzione diventa decisamente impraticabile quando si richiedono molte variabili: occorre usare array

- perché sono indicizzati
- perché posso popolarli/elaborarli con un ciclo

Con i vettori tipicamente il **for** risulta molto più comodo del **while** perché la variabile del ciclo viene usata per indicizzare gli elementi dell'array



## Esempio

Scrivere un programma che dichiara un array di dimensione 300 e scrive in ogni cella un numero da 1 a 300.

```
#include<stdio.h>
```



## Esempio

Scrivere un programma che dichiari un array di dimensione 300 e scrive in ogni cella un numero da 1 a 300.

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
    int vet[300]; int i;
```

```
    for (i = 0; i < 300 ; i++)
```

```
        vet[i] = i + 1 ;
```

```
}
```

Tipico uso del for per scorrere un array



## Il valore dell'array

Abbiamo visto che gli elementi dell'array contengono valori del tipo dell'array.

Quando scrivo

```
int vet[300],
```

So che in **vet[0]** troverò un intero.

Cosa c'è invece in **vet**?

- L'indirizzo del primo elemento in memoria, i.e.

```
vet == &vet[0];
```



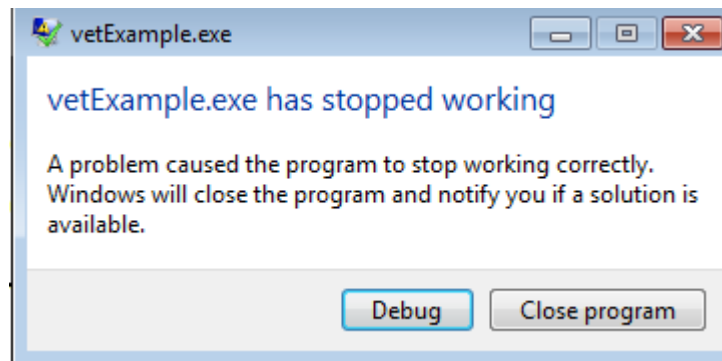
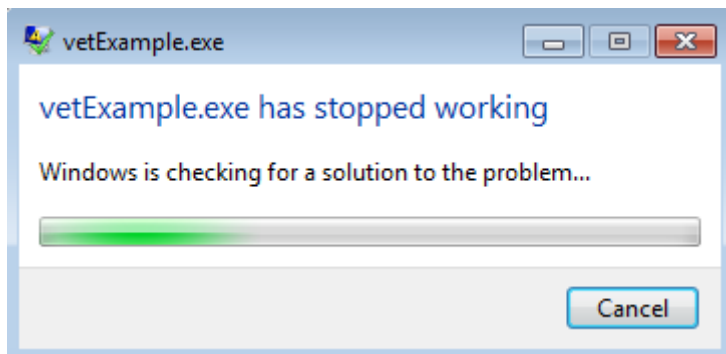
## Le Dimensioni degli Array

- Non è possibile accedere ad un elemento dell'array ad una posizione superiore alla dimensione:

```
int vet[20];
```

scrivere `vet[40]` (o anche solo `vet[20]` visto che le 20 celle vanno da `vet[0]` a `vet[19]`).

- In tal caso si ha **segmentation fault**, che nella migliore delle ipotesi si manifesta **solamente** a run-time (come quando si dimentica `&` in `scanf(...)`).





## Il Costruttore Array, COSE DA NON FARE!

*Errore*

```
int dim; /* il valore a dim è associato solo  
durante l'esecuzione */
```

```
scanf("%d", &dim);
```

```
X float resti[dim]; /* quindi il compilatore non  
sa quanto spazio riservare in memoria per  
resti */
```

Anche se il gcc lo permette, questo NON è da fare  
perché espone il codice a vulnerabilità



## #define

- Spesso si ricorre alla direttiva di precompilazione **define** per dichiarare un array

**#define** NOME\_DEFINE valoreNumerico

- Prima della compilazione, ogni istanza di **NOME\_DEFINE** (riferibile all'uso di variabile) verrà sostituita da **valoreNumerico**
- Se dichiaro **int vet[NOME\_DEFINE]** ; le dimensioni di **vet** sono note prima di iniziare la compilazione
- L'utilizzo di **define** rende il codice più leggibile, e facilmente modificabile quando occorre cambiare la dimensione dell'array (richiede comunque la ricompilazione del codice sorgente)
- **NB** non occorre il ; dopo **valoreNumerico**



## Esempio: Acquisizione di un array

Non esistono funzioni/comandi per acquisire un array di numeri (i.e., l'omologo di **scanf ("%d" . .)**)





## Esempio: Acquisizione di un array

Non esistono funzioni/comandi per acquisire un array di numeri (i.e., l'omologo di `scanf("%d" ..)`)

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i;
    for(i = 0; i < MAX_LEN; i++)
    {
        printf("Inserire elemento posizione %d" , i+1);
        scanf("%d" , &v1[i]);
    }
}
```



## Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i;
    for(i = 0; i < MAX_LEN; i++)
    {
        printf("Inserire elemento posizione %d" , i+1);
        scanf("%d" , &v1[i]);
    }
}
```

Uso **MAX\_LEN** come una costante nel codice



## Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i;
    for(i = 0; i < MAX_LEN; i++)
    {
        printf("Inserire elemento posizione %d" , i+1);
        scanf("%d" , &v1[i]);
    }
}
```

L'acquisizione con **scanf** avviene come per una qualsiasi variabile intera

Uso **MAX\_LEN** come una costante nel codice



## Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i;
    for(i = 0; i < MAX_LEN; i++)
    {
        printf("Inserire elemento posizione %d" , i+1);
        scanf("%d" , &v1[i]);
    }
}
```

Dettaglio per evitare di stampare «Inserire elemento posizione 0»

L'acquisizione con **scanf** avviene come per una qualsiasi variabile intera

Uso **MAX\_LEN** come una costante nel codice



## Le Dimensioni degli Array: Effettive vs Reali

- Distinguere tra dimensioni **reali** e dimensioni **effettive**
- Le dimensioni **reali** sono quelle con cui viene **dichiarato un array**. Sono fissate prima della compilazione, non modificabili. Si fissano «grandi a sufficienza»
- Le dimensioni **effettive** delimitano la **parte dell'array che si utilizzerà** durante l'esecuzione.
  - Possono essere specificate dall'utente in una variabile (previo controllo di compatibilità con quelle reali e.g., con **do while**)
- Esempio: modificare il programma precedente richiedendo prima all'utente quanti elementi inserire nell'array



## Le Dimensioni degli Array: Effettive vs Reali

Esempio: `int v1[11]` ; con dimensioni effettive  $n = 5$  ;

0
1
2
3
4
524
5455
431
987
745
65

Dimensioni effettive dell'array: le celle che vanno da 0 a n (specificato dall'utente in una variabile)

Dimensioni reali dell'array: definite da **MAX\_LEN**



## Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i, n; // n contiene le dimensioni effettive
do
{
    printf("quanti numeri vuoi inserire?");
    scanf("%d" , &n);
}while (n < 0 || n > MAX_LEN);

for (i = 0; i < n; i++)
{
    printf("Inserire elemento posizione %d" , i+1);
    scanf("%d" , &v1[i]);
}
}
```



## Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i, n; // n contiene le dimensioni effettive
    do
    {
        printf("quanti numeri vuoi inserire?");
        scanf("%d" , &n);
    } while (n < 0 || n > MAX_LEN);

    for(i = 0; i < n; i++)
    {
        printf("Inserire elemento posizione %d" , i+1);
        scanf("%d" , &v1[i]);
    }
}
```

Sono certo che **n** è compatibile con le dimensioni reali di **v1**





## Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN];
    int i, n; // n contiene le dimensioni effettive
do
{
    printf("quanti numeri vuoi inserire?");
    scanf("%d" , &n);
}while (n < 0 || n > MAX_LEN);

for (i = 0; i < n; i++)
{
    printf("Inserire elemento posizione %d" , i+1);
    scanf("%d" , &v1[i]);
}
}
```

Da qui in poi opero solo sulle prime **n** celle, quelle che vanno dall'indice 0 ad **n-1**



## Stampa dei valori dell'array

In generale non esiste un fattore di conversione per stampare gli array. Quindi occorre procedere iterando



## Stampa dei valori dell'array

In generale non esiste un fattore di conversione per stampare gli array. Quindi occorre procedere iterando

Assumiamo che l'array **v1** abbia dimensioni effettive **n**

```
printf("\nHai inserito: [");  
    for(i = 0 ; i <n ; i++)  
        printf(" %d ", v1[i]);  
    printf("]");
```

Per gli array di caratteri questo non è necessario (ma è comunque possibile seguire questa strada)



## Esempi

- Scrivere un programma che richiede all'utente di inserire una sequenza di numeri, specificando anticipatamente il numero di elementi che si intende inserire (controllare che sia compatibile con le dimensioni massime, 100)
- Il programma
  - Calcola il massimo della sequenza
  - Stampa tutti gli elementi inseriti in ordine contrario.
  - Stampa tutti i numeri dal massimo fino all'ultimo elemento dell'array
  - Conta le occorrenze del massimo



## Assegnamento tra array

- Non c'è un modo per direttamente assegnare **tutti** i valori in un primo array ad un secondo array

```
#include <stdio.h>
```


```
void main()
```

```
{   int v1[300], v2[300];
```

```
    int i;
```

```
    for(i = 0 ; i < 300 ; i++)
```

```
        v1[i] = i+1;
```

```
     v2 = v1;
```

```
}
```

**incompatible types  
when assigning to  
type 'int[300]' from  
type 'int \*'**



## Assegnamento tra array

Occorre operare su ogni singolo elemento dell'array!



## Assegnamento tra array

Occorre operare su ogni singolo elemento dell'array!

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN], v2 [MAX_LEN];
    int i;
    // popolo v1
    for(i = 0; i < MAX_LEN; i++)
        v1[i] = i;
    // copio i valori in v2
    for(i = 0; (i < MAX_LEN) ; i++)
        v2[i] = v1[i];
    // stamppo
    for(i = 0; (i < MAX_LEN) ; i++)
        printf("\nv1[%d] = %d , v2[%d] = %d", i,
            v1[i], i, v2[i]);
}
```



## Confronto tra array

Non c'è un modo per direttamente confrontare **tutti** i valori in due array

```
#include <stdio.h>
```

```
void main()
```

```
{   int v1[300], v2[300];
```

```
    int i;
```

```
    for(i = 0 ; i < 300 ; i++)
```

```
        { v1[i] = i+1;
```

```
          v2[i] = v1[i]; }
```

```
 if (v1 == v2)
```

```
    printf("ok"); }
```

non da  
errore di  
compilazione  
ma non fa  
quello che  
vorremmo...





## Confronto tra array

Occorre operare su **ogni singolo elemento!**



## Confronto tra array

Occorre operare su ogni singolo elemento: quindi

```
#define MAX_LEN 300
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN], v2 [MAX_LEN];
    int i, uguali;
    for(i = 0; i < MAX_LEN; i++)
    {
        v1[i] = i+1;
        v2[i] = v1[i];
    }
    uguali = 1;
    for(i = 0; (i < MAX_LEN) && uguali; i++)
        if(v1[i] != v2[i])
            uguali = 0;
    if(uguali)
        printf("ook!");}
```



## Confronto tra array

Occorre operare su ogni singolo elemento: quindi

```
#define MAX_LEN 300
#include <stdio.h>
void main()
{
    int v1 [MAX_LEN], v2 [MAX_LEN];
    int i, uguali;
    for(i = 0; i < MAX_LEN; i++)
    {
        v1[i] = i+1;
        v2[i] = v1[i];
    }
    uguali = 1;
    for(i = 0; (i < MAX_LEN) && uguali; i++)
        if(v1[i] != v2[i])
            uguali = 0;
    if(uguali)
        printf("ook!");
}
```

Variabile di flag, diventa 0 appena trova una cella per cui **v1** e **v2** differiscono

Scorro tutti gli elementi dei vettori. Mi arresto appena trovo due elementi diversi

Sta per (**uguali != 0**)



## Variabili di Flag per Verificare Condizioni su Array

Per controllare che una condizione (uguaglianza in questo caso) sia soddisfatta da tutti gli elementi del vettore

```
uguali = 1;
    for (i = 0; (i < MAX_LEN); i++)
        if (v1[i] != v2[i])
            uguali = 0;
```


Al termine del ciclo, se `uguali` è rimasta 1 sono certo che la condizione da verificare **non è mai stata negata** (i.e., `v1[i] != v2[i]` è sempre stata falsa). Quindi che **tutti** gli elementi degli array coincidono.

- La variabile di flag (`uguali`) può solo cambiare da **1** in **0**
- Ovviamente il ruolo di **0** e di **1** possono essere invertiti nel codice sopra



## Errore Frequente

Errore frequente: modificare il valore della variabile di flag nel  
anche nel verso opposto.

```
uguali = 1;
  for (i = 0; (i < MAX_LEN); i++)
    if (v1[i] != v2[i])
      uguali = 0;
       else
      uguali = 1;
```

Alla fine del ciclo se uguali è 1 posso solo concludere che l'ultima  
coppia di elementi controllati coincide!

Quando la condizione di permanenza nel ciclo è  
(i < MAX\_LEN) && uguali il problema non si pone ma  
else risulta comunque inutile



## Copiare alcuni elementi da un array ad un altro

- In molti casi è richiesto di **scorrere** un array **v1** e di **selezionare** alcuni valori secondo una data condizione.
- Tipicamente i valori selezionati in **v1** vengono **copiati in un secondo array, v2**, per poter essere utilizzati.
- È buona norma copiare i valori **nella prima parte di v2**, eseguendo quindi una copia «senza lasciare buchi».
- È anche necessario sapere quali sono i valori significativi in **v2** e quali no.

*Esempio* : copiare i numeri pari in **v1** in **v2**

■ <b>v1</b>	5	6	7	89	568	68	657	989	96	98
✗ <b>v2</b>	?	6	?	?	568	68	?	?	96	98
■ <b>v2</b>	6	568	68	96	98	?	?	?		



## Copiare alcuni elementi da un array ad un altro

Per fare questo è necessario usare **due** indici:

- **$i$**  per **scorrere  $v1$** : parte da **0** e arriva a  **$n1$** , la dimensione effettiva di  **$v1$** , con **incrementi regolari**.
- **$n2$**  parte da **0** e viene **incrementata solo quando un elemento viene copiato**.
  - **$n2$**  indica quindi il **primo elemento libero in  $v2$** ,
  - al termine,  **$n2$**  conterrà il **numero di elementi in  $v2$** , quindi la sua **dimensione effettiva**

5	6	7	89	568	68	657	989	96	98
---	---	---	----	-----	----	-----	-----	----	----

$i = 10;$   
 $n1 = 10;$

6	568	68	96	98	?	?	?
---	-----	----	----	----	---	---	---

$n2 = 5;$



## Esempio

Chiedere all'utente di inserire un array di interi (di dimensione definita precedentemente) e quindi un numero intero  $t$ . Il programma quindi:

- salva gli elementi inseriti in un vettore  $v1$ .
- Copia tutti gli elementi di  $v1$  che sono maggiori di  $n$  in un secondo vettore  $v2$ .
- La copia deve avvenire nella parte iniziale di  $v2$ , senza lasciare buchi.





## Esempio

```
printf("\nInserire la soglia");
scanf("%d" , &t);
n2 = 0;
for(i = 0; i < n1; i++)
if(v1[i] > t)
{
    v2[n2] = v1[i];
    n2++; //n2 è la prima posizione libera in v2
}
printf("\n Maggiori di %d sono: [" , t);
//n2 ora è la lunghezza effettiva di v2
for(i = 0 ; i <n2 ; i++)
    printf(" %d, ", v2[i]);
printf("]");
```



# Array di Caratteri

- Stringhe



## Array di Caratteri: le stringhe

- Nel C le stringhe (sequenze ininterrotte di caratteri) sono realizzate mediante array di caratteri

*Esempio*

```
char luogo[100];
```

è un array atto a contenere 100 elementi di tipo **char**

- Dato il frequente utilizzo ci sono standard e comandi particolari per facilitare l'uso delle stringhe,
  - I/O
  - Calcolo lunghezza
  - Confronto e Copia

**NB NON** esiste il tipo predefinito “string” né altri simili



## Acquisizione e Stampa di Stringhe

- Come per ogni array è possibile popolare un array di caratteri mediante inserimento carattere per carattere

```
printf("Inserire lunghezza stringa");  
scanf("%d" , &n);  
for(i = 0; i < n; i++)  
    scanf("%c" , &luogo[i]); fflush(stdin);
```

- Alternative più comode:
  1. `scanf("%s" , luogo);`
  2. `gets(luogo);`

NB `luogo` è l'indirizzo del primo elemento `&luogo[0]`, `scanf` quindi non ha bisogno della `&`.

NB `gets` richiede la libreria `string`, quindi occorre inserire `#include <string.h>`



## Esempio

Acquisizione e stampa di una stringa elemento per elemento



## Esempio

Acquisizione e stampa di una stringa elemento per elemento

```
void main()
{
  int i;
  char s[10];
  printf("\ninserire carattere 1");
  scanf("%c", &s[0]); scanf("%*c");
  for (i = 1; i < 10; i++)
    {printf("\ninserire carattere %d", i+1);
     scanf("%c", &s[i]); scanf("%*c");
    }
  for (i = 0; i < 10; i++)
    printf("%c\n", s[i]);
}
```

Oppure  
fflush(stdin)  
che funziona  
anche per le  
stringhe



## Esempio

Acquisizione e stampa di una stringa elemento per elemento

```
void main()
{
    int i;
    char s[10];
    printf("\ninserire carattere 1");
    scanf("%c", &s[0]); scanf("%*c");
    for (i = 1; i < 10; i++)
        {printf("\ninserire carattere %d", i+1);
        scanf("%c", &s[i]); scanf("%*c");
        }
    for (i = 0; i < 10; i++)
        printf("%c\n", s[i]);
}
```

E' necessario inserirlo anche qua, altrimenti si acquisisce solo l'invio (pulisco il carattere inserito)



## Esempio

Acquisizione e stampa di una stringa elemento per elemento

```
void main()
{
int i;
char s[10];
for (i = 0; i < 10; i++)
    {printf("\ninserire carattere %d", i+1);
    scanf("%c", &s[i]); scanf("%*c");
    }
for (i = 0; i < 10; i++)
    printf("%c\n", s[i]);
}
```

Andava bene  
anche così  
ovviamente





## Il Terminatore di Stringa

- Sia `scanf ("%s", ...)` che `gets (...)` delimitano la parte significativa (i caratteri inseriti dall'utente) con il carattere speciale `'\0'` (con codifica ASCII = 0).
  - Se a `gets (luogo)`; digito: Milano in memoria verrà scritto `"Milano\0"`.
  - Differenze
    - `scanf ("%s", luogo)`; termina l'inserimento al primo spazio o invio.
    - `gets (luogo)`; termina l'inserimento al primo invio
- Ecco cosa acquisiscono se digito: Piazza san Babila
- `scanf ("%s", luogo)`; `"Piazza\0"`
  - `gets (luogo)`; `"Piazza san Babila\0"`



## Stampa di Stringhe

- È possibile stampare i caratteri in una stringa fino al terminatore utilizzando `printf("%s", ...)` ;

*Esempio*

```
gets (luogo) ;  
printf("Io abito a %s", luogo) ;
```

- Quando si popola la stringa carattere per carattere, è necessario inserire il terminatore di stringa `'\0'`

```
char provincia[3] ;  
provincia[0] = 'M' ;  
provincia[1] = 'I' ;  
provincia[2] = '\0' ;  
printf("Io abito a %s", provincia) ;
```

Occorre tener conto del terminatore anche nella dimensione



## Calcolo della Lunghezza

- È possibile calcolare la lunghezza di una stringa andando a contare gli elementi che precedono `'\0'`.

```
int len = 0;
char luogo[100];
scanf("%s", luogo);
while(luogo[len] != '\0')
    len++;
printf("%s e' lunga %d", luogo, len);
```

La lunghezza di una stringa corrisponde alla posizione del carattere `'\0'`. Il valore viene assegnato a `len`

- Oppure è possibile usare la funzione `strlen`, contenuta nella libreria `string`  
`len = strlen(luogo);`



## Esempio (Calcolo Lunghezza di Una Stringa)

```
#include<string.h>
void main()
{
    int coincidono, len1, len2, flag;
    char str1[30], str2[30], str3[30];
    printf("inserire prima stringa ");
    gets(str1);
    printf("inserire seconda stringa ");
    gets(str2);

    // calcolo le lunghezze
    len1 = strlen(str1);
    // calcolo le lunghezze
    len2 = strlen(str2);

    printf("\n%s e' lunga %d, %s e' lunga %d", str1,
len1, str2, len2);
}
```



## Esempio (Calcolo Lunghezza di Una Stringa)

```
#include<string.h>
void main()
{
    int coincide;
    char str1[30];
    printf("inserire prima stringa pollo\n");
    gets(str1);
    printf("inserire seconda stringa cane\n");
    gets(str2);

    // calcolo le lunghezze
    len1 = strlen(str1);
    // calcolo le lunghezze
    len2 = strlen(str2);

    printf("\n%s e' lunga %d, %s e' lunga %d", str1,
len1, str2, len2);
}
```

"E:\My Documents\Google Drive\poli\Didattica\2012\_InfoB\_En

```
inserire prima stringa pollo
inserire seconda stringa cane
```

```
pollo e' lunga 5, cane e' lunga 4
```



## Confronto tra Stringhe

È possibile verificare se due stringhe coincidono:

1. Verificando che la loro lunghezza coincide **&&**
2. Verificando se esse coincidono in ogni elemento



## Confronto tra Stringhe

È possibile verificare se due stringhe coincidono:

1. Verificando che la loro lunghezza coincide **&&**
2. Verificando se esse coincidono in ogni elemento

```
int flag = 1; int len, i;
char str1[30], str2[30];
gets(str1); gets(str2);
len = strlen(str1);
if(len == strlen(str2))
    for(i = 0; i < len && flag; i++)
        { if(str1[i] != str2[i])
            flag = 0; }
else // non hanno la stessa lunghezza
    flag = 0;
printf("%s == %s : %d", str1, str2, flag);
```

È indispensabile mettere le parentesi attorno a questo if altrimenti l'else seguente viene associato a questo e non a quello più esterno



## Confronto tra Stringhe

- Oppure è possibile usare la funzione **strcmp**, contenuta nella libreria **string**. Sintassi

```
strcmp(s1 , s2) ;
```

- Diventa un intero
  - == 0 se coincidono
  - < 0 se **s1** precede **s2** in ordine alfabetico
  - > 0 se **s1** segue **s2** in ordine alfabetico

```
int cmpr; cmpr = strcmp(str1 , str2) ;
```

```
if (cmpr == 0)
```

```
    printf("%s e %s coincidono", str1, str2) ;
```

**NB.** Le stringhe **str1** e **str2** devono terminare con `'\0'`





## Esempio di Confronto Tra Stringhe

```
#include<string.h>
void main()
{
int coincidono, len1, len2, flag;
char str1[30], str2[30], str3[30];
...
// strcmp che restituisce 0 se coincidono
flag = strcmp(str1 , str2);
// metto coincidono a 1 quando flag è 0
coincidono = (flag == 0);
printf("\n%s == %s : %d", str1, str2, coincidono);
if (flag > 0)
    printf("\n%s precede%s (flag = %d)", str2, str1, flag);
if(flag < 0)
    printf("\n%s precede%s (flag = %d)", str1, str1, flag);
}
```



## Esempio

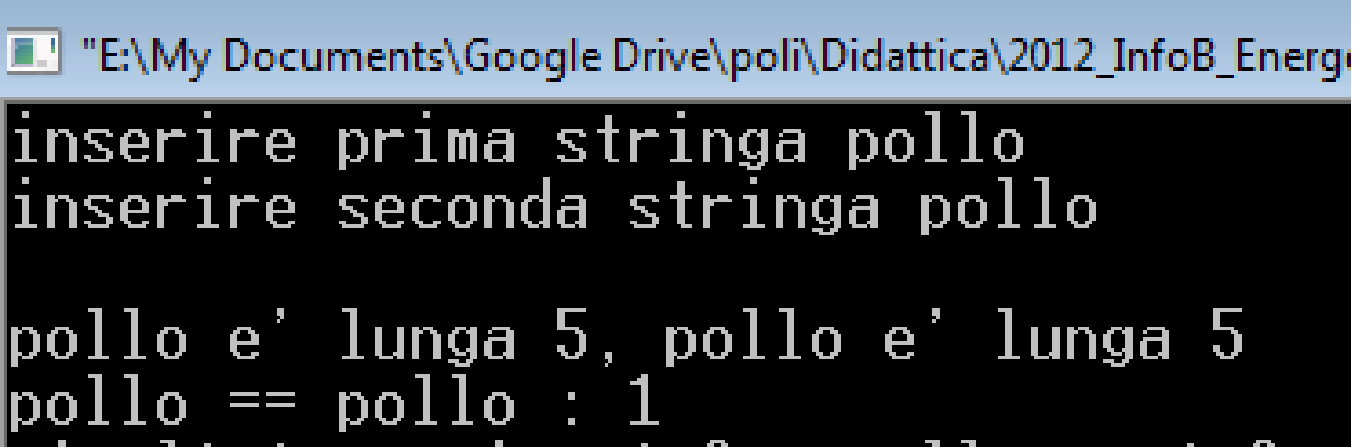
```
#include <string.h>
void main()
{
    int coincidono;
    char str1[30];
    ...
    // strcmp compare stringhe
    flag = strcmp(str1, str2);
    // metto coincidono a 1 quando flag è 0
    coincidono = (flag == 0);
    printf("\n%s == %s : %d", str1, str2, coincidono);
    if (flag > 0)
        printf("\n%s precede%s (flag = %d)", str2, str1, flag);
    if (flag < 0)
        printf("\n%s precede%s (flag = %d)", str1, str1, flag);
}
```

"E:\My Documents\Google Drive\poli\Didattica\2012\_InfoB\_Energetici\_e\_I  
inserire prima stringa pollo  
inserire seconda stringa cane  
pollo e' lunga 5, cane e' lunga 4  
pollo == cane : 0  
cane viene prima di pollo (flag = 1)



## Esempio

```
#include <string.h>
void main()
{
int coincidono;
char str1[30];
char str2[30];
// strcpy che restituisce 0 se coincidono e non 0 se no
flag = strcmp(str1 , str2);
// metto coincidono a 1 quando flag è 0
coincidono = (flag == 0);
printf("\n%s == %s : %d", str1, str2, coincidono);
if (flag > 0)
    printf("\n%s precede%s (flag = %d)", str2, str1, flag);
if (flag < 0)
    printf("\n%s precede%s (flag = %d)", str1, str1, flag);
}
```





## Esempio

```
#include <string.h>
void main()
{
    int coincidono;
    char str1[30];
    char str2[30];
    // inserisci la prima stringa
    // inserisci la seconda stringa
    // elefante e' lunga 8, struzzo e' lunga 7
    // elefante == struzzo : 0
    // elefante viene prima di struzzo (flag = -1)
    // strcmp che restituisce 0 se coincidono
    flag = strcmp(str1, str2);
    // metto coincidono a 1 quando flag è 0
    coincidono = (flag == 0);
    printf("\n%s == %s : %d", str1, str2, coincidono);
    if (flag > 0)
        printf("\n%s precede %s (flag = %d)", str2, str1, flag);
    if (flag < 0)
        printf("\n%s precede %s (flag = %d)", str1, str2, flag);
}
```

"E:\My Documents\Google Drive\poli\Didattica\2012\_InfoB\_Energetici\_e\_Meccanica

inserire prima stringa elefante  
inserire seconda stringa struzzo

elefante e' lunga 8, struzzo e' lunga 7  
elefante == struzzo : 0  
elefante viene prima di struzzo (flag = -1)

strcmp che restituisce 0 se coincidono




## Copia tra Stringhe

- È possibile eseguire la copia elemento per elemento da un array ad un altro, come nell'esercizio precedente
- Oppure è possibile usare la funzione **strcpy**, contenuta nella libreria **string**. Sintassi:

```
strcpy(s1 , s2) ;
```



- Copia il contenuto di **s2** in **s1** incluso il `'\0'`
  - Per accodare le stringhe si usa la funzione **strcat**, contenuta nella libreria **string**. Sintassi:
- ```
strcat(s1 , s2) ;
```
- 
- Accoda il di **s2** in **s1** (il `'\0'` appare solo alla fine)



## Esempio di Copia Tra Stringhe

```
#include<string.h>
void main()
{
int coincidono, len1, len2, flag;
char str1[30], str2[30], str3[30];
...

// copia in str3 il contenuto di str2
strcpy(str3, str2);
printf("\nrisultato copia str2 = %s e str3 =%s",
str2, str3);

// accoda
strcat(str3, str1);
printf("\naccodo str1 a str3: %s ", str3);
}
```



## Esempio

```
#include <string.h>
void main()
{
int coincido;
char str1[30];
...

// copia in str2
strcpy(str2, str1);
printf("\nrisultato copia str2 = %s e str3 =%s",
str2, str3);

// accoda
strcat(str3, str1);
printf("\naccodo str1 a str3: %s ", str3);
}
```

```
"E:\My Documents\Google Drive\poli\Didattica\2012_InfoB_Energetici_e_M
inserire prima stringa pollo
inserire seconda stringa cane
pollo e' lunga 5, cane e' lunga 4
pollo == cane : 0
cane viene prima di pollo (flag = 1)
risultato copia str2 = cane e str3 =cane
accodo str1 a str3: canepollo
```



## TODO: Esercizio su switch e array

Scrivere un programma che acquisisce un vettore di interi definito dall'utente e quindi calcola

- Il numero di multipli di 8
- Il numero di multipli di 4
- Il numero di multipli di 2

Presenti nel vettore e quindi stampa un istogramma (verticale) per visualizzare le occorrenze

- *Es* su [8 12 1 3 2]

```
          *
        *  *
       *  *  *
      -  -  -
     8   4   2
```