

**ATTENZIONE! QUESTO MATERIALE DIDATTICO È PER USO PERSONALE DELLO STUDENTE  
ED È COPERTO DA COPYRIGHT. NE È VIETATA LA RIPRODUZIONE O IL RIUTILIZZO ANCHE PARZIALE,  
AI SENSI E PER GLI EFFETTI DELLA LEGGE SUL DIRITTO D'AUTORE.**

---

*Stringhe*

# Stringhe

- Finora sappiamo leggere, scrivere, elaborare singoli caratteri
- Avremmo bisogno anche di elaborare parole, frasi, ...
- In C, si possono usare le *stringhe*, che sono un caso particolare di *array di caratteri*.
- Per quello che sappiamo finora, possiamo leggere un array di caratteri uno alla volta:

```
int i;
char s[10];
for (i=0; i<10; i++)
    scanf("%c", &s[i]);
```

s | f | e | r | r | a | r | a | 1 | 2 | 3 |

0 1 2 9

- e stamparlo con

```
for (i=0; i<10; i++)
    printf("%c", s[i]);
```

ferrara 123

f e r r a r a 1 2 3

- Però così posso leggere/scrivere solo un numero predefinito di caratteri

# Array di caratteri

lung  
7

- Se non so quanti sono a priori i caratteri, potrei usare una variabile per contenere la lunghezza

```
int lung, i;
```

```
char s[100];
```

s

f	e	r	r	a	r	a													
0	1	2	3	4	5	6	7										98	99	

```
printf("Immetti il numero di caratteri:");
```

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

```
printf("immetti i caratteri");
```

i

0	1
---	---

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

```
    scanf("%c", &s[i]);
```

f e r r a r a

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

```
    printf("%c", s[i]);
```

# Array di caratteri

- Oppure potrei usare un carattere speciale per indicare che i caratteri significativi finiscono lì

```
int i;
char s[100];
printf("immetti i caratteri (@ per terminare):");
```

@  
TERMINATORE

s

F	e	r	r	a	r	a	@												
0	1	2	3	4	5	6	7	8	9										99

```
i=-1;
do
{
    i++;
    scanf("%c",&s[i]);
} while (s[i]!='@');
i=0;
while (s[i]!='@')
{
    printf("%c",s[i]);
    i++;
}
```

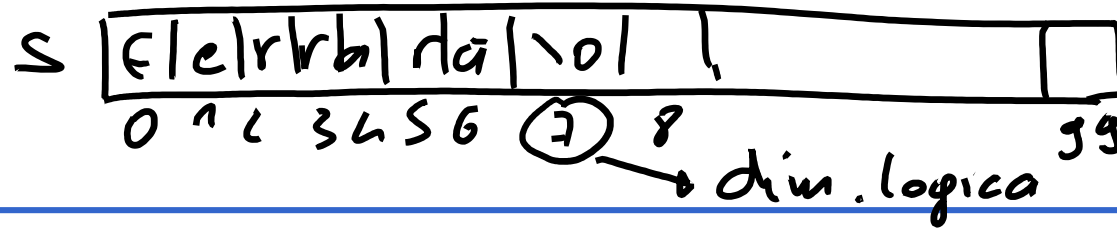
ferrara@

i

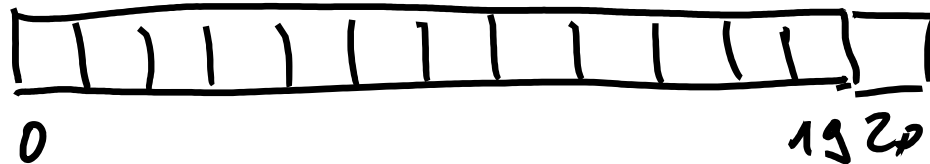
<del>X</del>	<del>0</del>	<del>X</del>	<del>2</del>	...	<del>6</del>	<del>7</del>	<del>0</del>	<del>2</del>	<del>2</del>	...	<del>6</del>	<del>7</del>
--------------	--------------	--------------	--------------	-----	--------------	--------------	--------------	--------------	--------------	-----	--------------	--------------

Ferrara

# Stringhe



- Nel linguaggio C si utilizza questa seconda visione.
- Viene utilizzato un **codice speciale** che non può comparire in nessuna stringa: il carattere con **codice ASCII 0**, indicato anche con:  $'\backslash 0' \neq '0' = 48$
- Come vedremo, se utilizziamo questa convenzione, il C ci fornisce alcune istruzioni comode già fatte (non dobbiamo ricostruirle noi).

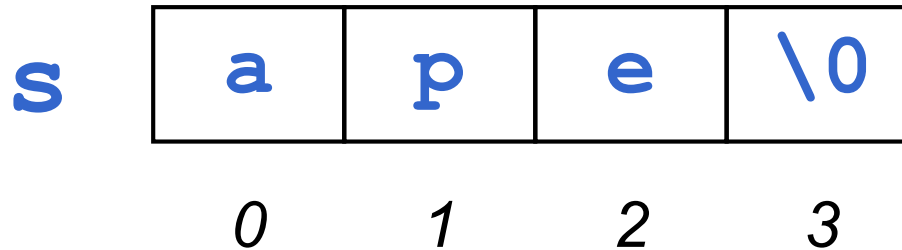


0 ← NULL-terminated strings

## STRINGHE: ARRAY DI CARATTERI

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

`char s[4];`

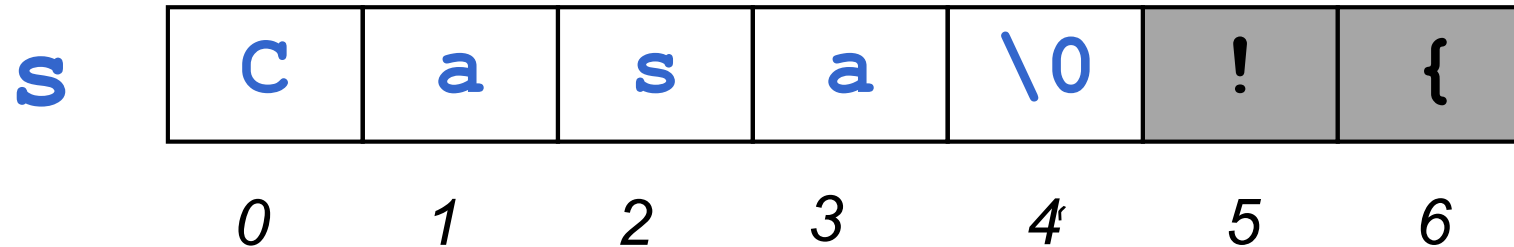


dimensione logica < dim. Fisica

- Un vettore di  $N$  caratteri può dunque ospitare stringhe lunghe al più  $N-1$  caratteri, perché una cella è destinata al terminatore '`\0`'.

# STRINGHE: ARRAY DI CARATTERI

- Un array di  $N$  caratteri può essere usato per memorizzare stringhe più corte `char s[7];`



- In questo caso, le celle oltre la  $k$ -esima ( $k$  essendo la lunghezza della stringa) sono concettualmente vuote: *praticamente sono inutilizzate e contengono un valore non significativo.*

`int a[4] = {2, 5, 1, 4};`

## STRINGHE: Inizializzazione

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

`s`

a	p	e	\0
0	1	2	3

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

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

`char s[4] = "ape";`  
`= "ferrara"`

a	p	e	\0
0	1	2	3

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

Attenzione alla lunghezza!



# 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';
```

INPUT a p e \

str [a|p|e|\0] i [0 1 2 3]

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

```
...
char str10[4];
scanf("%s", str);
```

str [a|p|e|\0] [ ] [ ]

0 1 2 3 4 ... 9

Per motivi che studieremo più avanti, nella scanf non si usa la & per le stringhe

Quando voglio parlare della stringa nella sua totalità non metto le parentesi quadre

printf("%s", str);

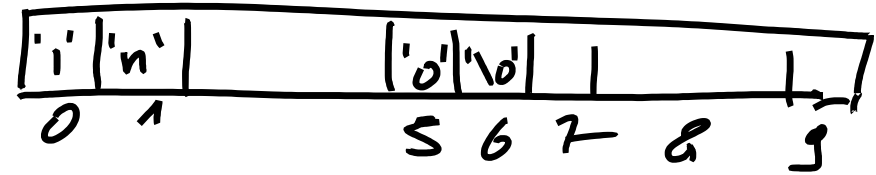
# Costanti di tipo stringa

- In generale, le doppie virgolette rappresentano le *costanti di tipo stringa*

```
main()
```

```
{ char s[10]="inizio";  
  printf("%s %s", s, "fine");  
}
```

OUTPUT: inizio \_ fine



ASSEGNAZIONE  
S = "fine";

ERRORE

Sono quindi delle sequenze di caratteri con il terminatore in fondo

## ESEMPIO

'A' < 'a'  
65 97

"A bc"  
precede  
"a bc"

"123"  
precede  
"abc"

### Problema:

*Date due stringhe di caratteri, decidere quale precede l'altra in ordine alfabetico.*

lessicografico

### Rappresentazione dell'informazione:

- poiché vi possono essere tre risultati ( $s1 < s2$ ,  $s1 == s2$ ,  $s2 < s1$ ), un boolean non basta
  - possiamo usare:
    - due boolean (uguale e precede)
    - tre boolean (uguale,  $s1 < s2$ ,  $s2 < s1$ )
    - un intero (negativo, zero, positivo)
- scegliamo la terza via.

"Mania"  
precede  
"Marta"

# CONFRONTO FRA STRINGHE


## 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];
    .....
}
```

$s1[i] \neq '\backslash 0'$   
 $s1[i] \neq 0$   
 $s1[i]$

negativo  $\leftrightarrow$  s1 precede s2  
 positivo  $\leftrightarrow$  s2 precede s1  
 zero  $\leftrightarrow$  s1 è uguale a s2

# L'operatore [ ]

- Le parentesi quadre si usano per
  - dichiarare (o definire) una variabile array:  
`char s[10];`
  - identificare un elemento di una variabile array  
`printf("%c", s[3]);`  

- Ovvero:
  - in fase di *dichiarazione*, mi dicono che quella variabile è un array
  - in fase di *utilizzo*, mi dicono di prendere un certo elemento dell'array (e non tutto l'array)
- Se voglio *utilizzare* l'intero array, non metto le parentesi quadre  
`printf("%s", s);`

# Stringhe e caratteri

## Caratteri

- *definizione:*  
`char c;`
- *contenuto:*  
*un carattere*
- *costante:*  
*un carattere fra apici singoli: 'a'*
- *codice nelle stringhe formato: %c*

## Stringhe

- *definizione:* *dim. fis.*  
`char s[10];`
- *contenuto:*  
*una sequenza di caratteri, terminata dal carattere '\0' ☺*
- *costante:*  
*una sequenza di caratteri fra apici doppi: "ciao"*
- *Codice nelle stringhe formato: %s*

# Esercizio

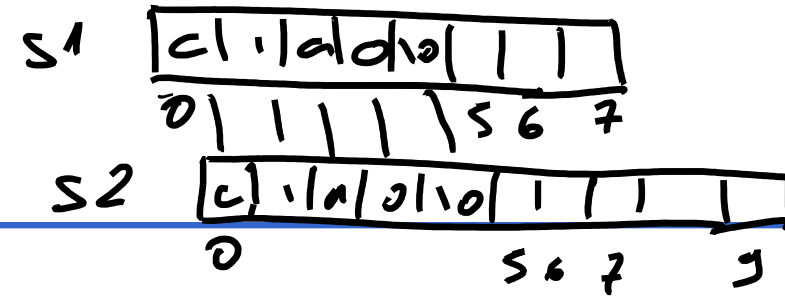
- Dire se le seguenti dichiarazioni sono corrette

- `char a='a';`
- `char b="ciao";` *WARNING*
- `char s[]='a';` *ERROR*
- `char t[]="ciao";`
- `char v[10]="ciao";`

- Dire se le seguenti istruzioni sono corrette, sapendo che vale la dichiarazione: `char c, s[5], t[5];`

- |  |  |
|--|--|
| • <code>printf("%c", c);</code> <i>ok</i>  | • <code>printf("%c", s[5]);</code> <i>SAECLVTO</i> |
| • <code>printf("%s", s);</code> <i>ok</i>  | • <code>printf("%s", s[5]);</code> <i>w</i>        |
| • <code>printf("%c", s);</code> <i>w</i>   | • <code>printf("%c", s[0]);</code> <i>ok</i>       |
| • <code>printf("%s", c);</code> <i>w</i>   | • <code>printf("%s", s[0]);</code> <i>w</i>        |
| • <code>s[5]=t[5];</code>  | • <code>c = s;</code> <i>w</i>                     |
| <div style="display: flex; justify-content: space-around; width: 100px;"> <span>↑</span> <span>↑</span> </div> <i>SAECLVTO</i> | • <code>c = s[2];</code> <i>ok</i>                 |

- Nelle istruzioni scorrette, si dica se si ha errore di compilazione



## ESEMPIO

### Problema:

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

### Ipotesi:

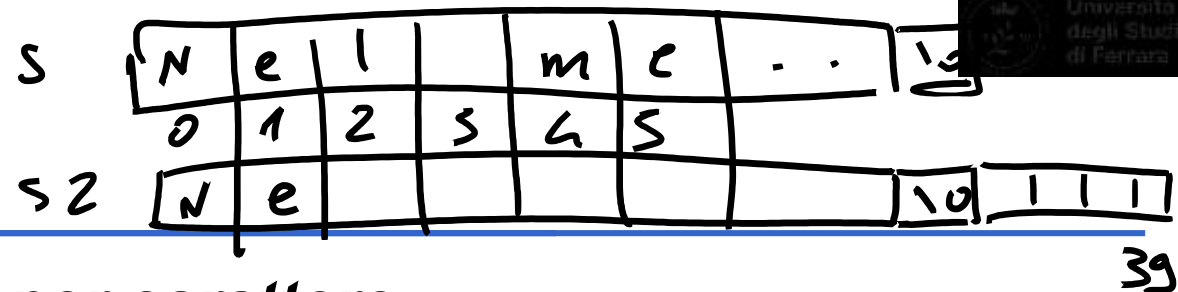
La stringa è “ben formata”, ossia correttamente terminata dal carattere '`\0`'.

### Specifica:

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



## ESEMPIO



Codifica: copia della stringa carattere per carattere

```
main()
```

```
{char s[] = "Nel mezzo del cammin di";
```

```
char s2[40];
```

La dimensione deve essere tale da garantire che la stringa non ecceda

```
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

```
int a = 4, b;  
b = a;
```

### Perché non fare così?

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

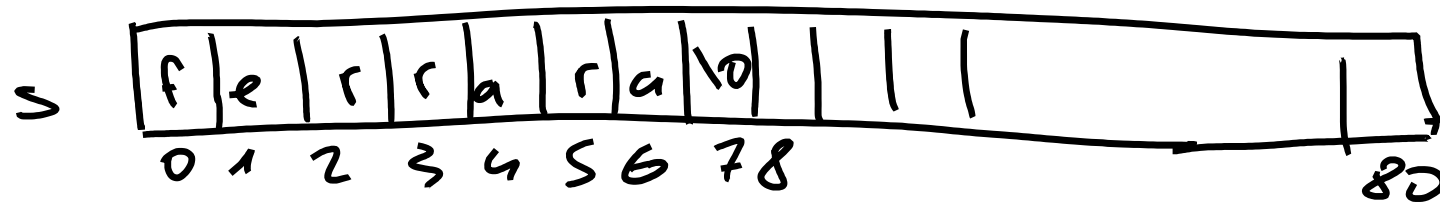
ERRORE DI COMPILAZIONE:  
*left operand must be l-value*

**PERCHÉ GLI ARRAY NON POSSONO  
ESSERE MANIPOLATI NELLA LORO INTEREZZA !**

Vedremo più avanti il perché

# Esercizio

Leggere da tastiera una stringa e dire qual è la sua lunghezza



`char s[81];`

INPUT ferrara

~~for~~

`i = 0;`

`while (s[i] != '\0') {`

`i++;`

~~for~~

`}`

`i` contiene la lunghezza

# Esercizio

*Date due stringhe, mettere in una terza stringa la concatenazione delle due*

- *Es: char a[]="gian", b[]="luca", c[20];  
mettere nella stringa c: "gianluca")*

a 

g	i	a	n	\0
---	---	---	---	----

  
0 1 2 3 4

b 

l	u	c	a	\0
---	---	---	---	----

  
0 1 2 3 4

c 

g	i	a	n		l	u	c	a	\0										
---	---	---	---	--	---	---	---	---	----	--	--	--	--	--	--	--	--	--	--

  
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

# Esercizio

Letti una stringa ed un carattere, verificare se il carattere compare nella stringa

i 0 s f|e|r|r|a|r|a|\0 c e  
0 1 2 3 4 5 6 7

```
i = 0;
while ( s[i] != '\0' && c != s[i] )
    i++;
if ( s[i] == '\0' )
    printf("Non trovato");
else
    printf("Trovato");
```



valigia  $\swarrow$  valigie  
 $\searrow$  valige

# Esercizio

- Il plurale di una parola che termina in «cia» (risp. «gia») termina in
  - «cie» (risp. «gie») se nel singolare «cia» (risp «gia») è preceduto da una vocale
  - «ce» (risp. «ge») altrimenti.
- Ad esempio: «valigia» → «valigie», «saggia» → «sagge», «camicia» → «camicie», «faccia» → «facce»    fascia → fasce
- Scrivere un programma che stampi il plurale di una parola che termina in «cia» o «gia» digitata dall'utente.

INPUT

OUTPUT

fascia

fasce

valigia

valigie

# Libreria sulle stringhe: *strcpy* e *strcat*

- Il C ha una libreria sulle stringhe:

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

- Alcune istruzioni utili:

- strcpy (Destinazione, Sorgente)**

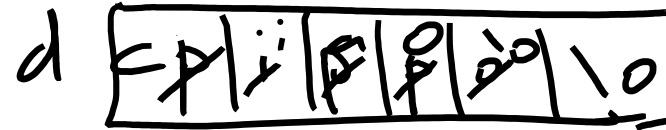
- copia la stringa Sorgente sulla Destinazione

- es:

```
char d[6]="pippo", s[5]="ciao";
```

```
strcpy(d,s);
```

```
printf("%s",d); ➡ stampa "ciao"
```



- strcat (Destinazione, Sorgente)**

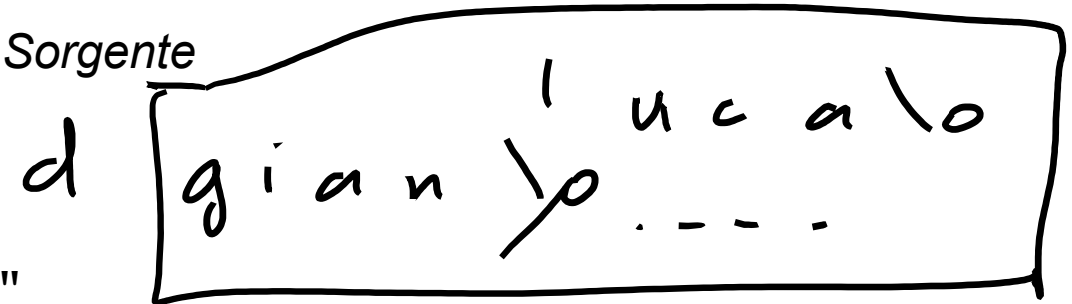
- aggiunge in fondo alla stringa Destinazione la Sorgente

- es:

```
char d[20]="gian", s[]="luca";
```

```
strcat(d,s);
```

```
printf("%s",d); ➡ stampa "gianluca"
```





# string.h: strcmp

- `strcmp(Stringa1, Stringa2)`
- *fornisce un valore*
  - $=0$  se le due stringhe sono uguali
  - $<0$  se *Stringa1* viene prima di *Stringa2* in ordine alfabetico
  - $>0$  se *Stringa2* viene prima di *Stringa1*
- *Es*

```
char s1[10], s2[10]; int diverse;
scanf("%s", s1);
scanf("%s", s2);
• diverse = .strcmp(s1, s2);
  if (diverse != 0)
    if (diverse > 0)
      printf("%s precede %s", s2, s1);
    else printf("%s precede %s", s1, s2);
  else printf("sono uguali");
```

# *string.h: strlen*

- <sup>int</sup>strlen(Stringa)
- *fornisce la lunghezza della Stringa*

strlen("ciao") → 4