Stringhe

Oct. 12

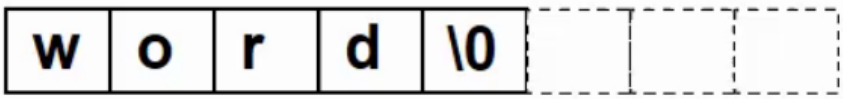
Char → occupa 1 byte.

💻 **stringhe** → definite come array di caratteri. Rappresentano dei caratteri da leggersi in fila. Le stringhe **NON** son un tipo di dato base. Esiste la libreria *#include <string.h>* per accedere a una serie di funzioni.

Dichiarazione di una stringa

```
char stringa[] = "word";
```

Il carattere **nullo** **'\0'** **termina** le stringhe. Perciò l'array stringa[] ha **5** elementi, (non *4*):



Può essere dichiarata anche come

```
char stringa[] = {'w', 'o', 'r', 'd', '\0'};//dobbiamo inserire il terminatore
```

Il carattere singolo è fra apici singoli 'w'

La parola, sequenza di caratteri è fra **apici doppi** "word" (il terminatore è inserito in automatico).

```
char str1[32]; //str1 ha spazio per 31 caratteri (da 0 a 30) + 1 per il terminatore \0 - (il 31 esimo elemento)
char str2[64]; //str2 ha spazio per 64 caratteri (da 0 a 63), il 63 esimo char è \0
```

Operazioni tra caratteri

```
c1 = 'b'; //Codice ascii di b - 8bit
c2 = 'c'; //codice ascii di c - 8 bit
```

```
c3 = c2 - c1; //Differenza tra i codici ASCII di c2 e c1
//Lo utilizziamo per capire quanti caratteri possono stare in mezzo a c2 e c1
```

Funzioni libreria <string.h>

▼ **strcpy**(str1, "ciao");

str1 avrà il valore di "ciao"

```
#include <string.h>
char str1[32]; //Dichiaro l'array e riservo lo spazio in memoria, ma è vuota.
char str2[64];
//Inizializza str1 con la stringa "alfa"

strcpy(str1, "alfa"); //Funzione copia dalla stringa "alfa" alla str1
strcpy(str2, str1); //copio il valore di str1 in str2
```

ATTENZIONE → **NON** si può fare str1 = "alfa", è errore

▼ x = **strlen**(str1)

conta i caratteri (tranne '\0') che sono presenti nella stringa.

```
//Se l'array è di 32 caratteri, ma ha solo 4 caratteri al suo interno, x sarà uguale a 4
x = strlen(str1); //quanti sono i caratteri presenti in str1 senza il carattere terminatore \0
```

▼ **printf**("%s", str1);

Stampa la stringa

```
printf("%s", str1); //stampa tutti i caratteri finchè non compare il carattere terminatore
```

▼ **scanf**("%s", str1);

Legge la stringa

```
scanf("%s", str1); //Siccome è un array, non c'è bisogno di mettere la &
```

Quando si fa una scan di un valore **scalare** si mette la &, altrimenti no

Esempio

```
char str1[32];
char str2[64];


scanf("%s", str1); //str1 contiene ciao (messo da tastiera)

strcpy(str2, str1); //str2 riceve il valore ciao
val = strlen(str2); //val assume il valore 4 (ciao è di 4 caratteri)

printf("%s\n", str2); //stampa ciao e va a capo
```

strlen("") ha il valore di 0. (viene contato solo il terminatore \0)

```
char stringa[] = "word"; //è composto da 5 caratter w o r d \0
stringa [3]; //è un'espressione di valore 'd'
```

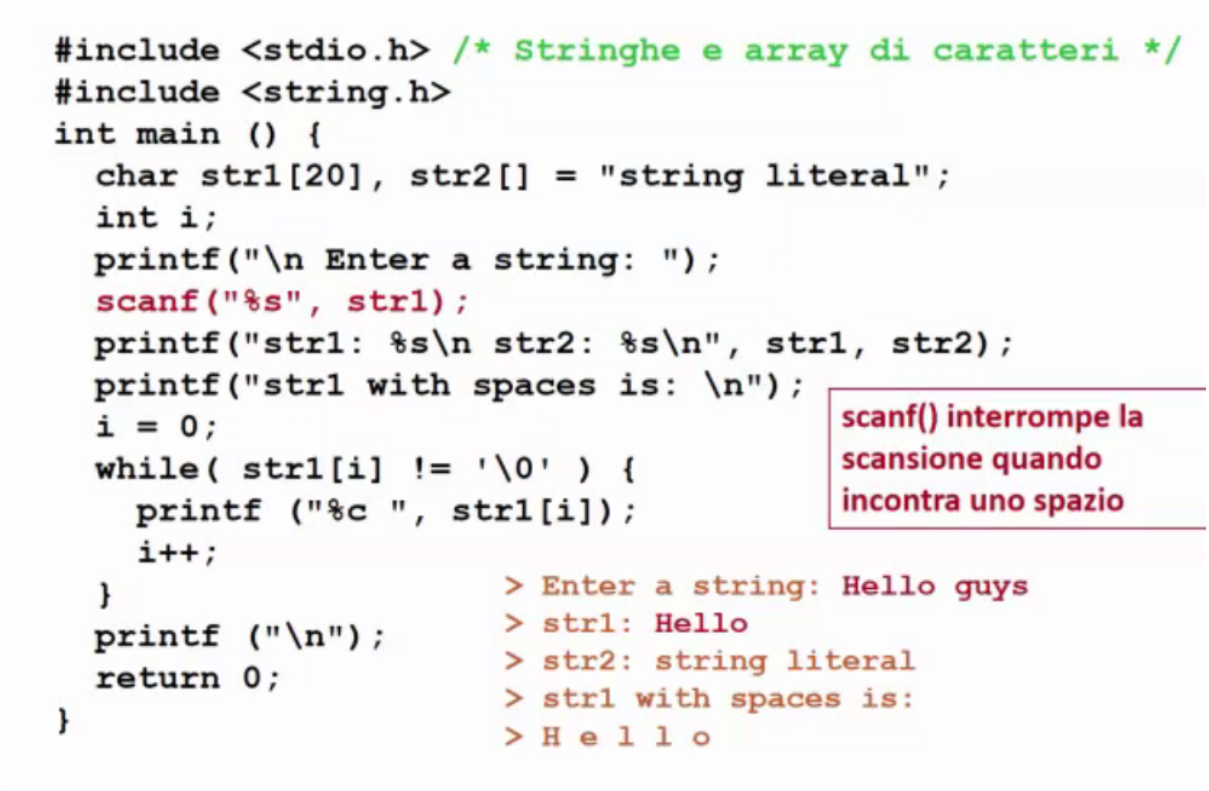
Il **nome** dell'array rappresenta **l'indirizzo del suo primo elemento**, perciò quando ci si vuole riferire all'intero array nella **scanf** non si mette il simbolo &

```
scanf("%s", stringa);
```

Questa scanf legge in input i caratteri fino a quando trova il carattere "blank" (spazio) o l'invio.

Lo spazio viene contato, ma possiamo inserirli solo nella dichiarazione della variabile, **NON** nella *scanf*.

Stampare una stringa con gli spazi



guys non viene stampato perchè durante l'acquisizione è stato inserito uno spazio tra "Hello" e "guys". Dovremmo utilizzare la funzione *gets()* se vogliamo inserire degli spazi nell'acquisizione della stringa.

Se non ci fosse la strcopy()?

Copieremmo un carattere alla volta da s2 a s1.

```
char s1[N], s2[N];
/*Assegnamento di s2 omesso*/
int i = 0;
while (i<=strlen(s2) && i<N){
  s1[i] = s2[i];
  i++;
}
```

NB: **N** è la lunghezza massima che le stringhe possono ricevere. strlen() è la lunghezza effettiva (quanti caratteri ci sono dentro alla stringa in quel momento). s1 deve essere sufficientemente grande da contenere s2.

Cosa stampano le seguenti printf?

```
int a = 2;
char amac[6] = "amac"; //utilizzati 4 caratteri + il tappo \0

amac[strlen(amac)] = amac[a]; //prende il 3 carattere di amac, si conta da 0 e a vale 2
//inserisce nella posizione 4 della stringa amac, il valore a. Si avrà quindi la parola amaca\0
printf("%s\n", amac);  //stamperà amaca

printf("%d\n", strlen(amac)); //sarà 5
```

Ogni char occupa 1 byte. 5 caratteri occupano 5 byte.

amac[strlen(amac)] = amac[a]; → prende il 3 carattere di amac, (si conta da 0 e la variabile **a** vale 2).
Quest'istruzione, inserisce nella posizione **4** (0, 1, 2, 3, 4), quindi come quinto carattere della stringa **amac**, il carattere **a** *(terzo carattere della stringa amac)*. Si avrà quindi la parola amaca\0 → l'array amac[6] ha dimensione 6. (posizioni da 0 a 5).

Confrontare due stringhe

▼ **strcmp(s1, s2)**

- **Restituisce** un intero
- C**onfronta** i codici ASCII delle due stringhe fino al '\0'

```
char s1[32], s2[64]; //Confronterà solo i 32 caratteri di s1 con i primi 32 caratteri di s2
int diverse;

diverse = strcmp(s1, s2);
if (diverse == 0)
  printf("Le stringhe sono UGUALI\n");
else if (diverse < 0)
  printf("%s PRECEDE %s", s1, s2);
else
  printf("%s SEGUE %s", s1, s2);
```

Esempio comparazione

```
#include <stdio.h>
#include <string.h>
#define DIM 3

int main()
{
  int diverse;
  char str1[32] = "casa", str2[64] = "casona";

  diverse = strcmp(s1,s2);
}
```

In questo caso siccome diverse è < 0, **str1 precede str2**

Se non ci fosse la funzione della libreria <string.h> (strcmp), dovremmo fare come segue:

```
char s1[N], s2[M];
int i = 0; ...
while( i<strlen(s1) && i<strlen(s2)
                   && s1[i] == s2[i] ) i++;
if (s1[i] == s2[i]) /* s1[i] == '\0'*/
    ... Le stringhe sono Uguali ...
```