

Tipi strutturati in C

Stringhe

Stringhe

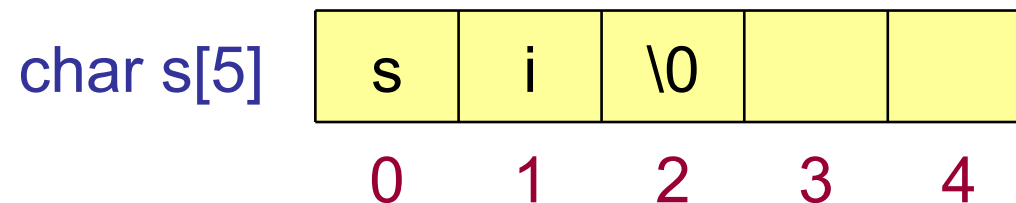
- Una **stringa di caratteri** in C è un array di caratteri *terminato dal carattere '\0'*
- Un vettore di N caratteri può dunque ospitare stringhe *lunghe al più N-1 caratteri*, perché una cella è destinata al **terminatore '\0'**

char s[5]

C	i	a	o	\0
0	1	2	3	4

Stringhe

- 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* (k essendo 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[5] = {'C', 'i', 'a', 'o', '\0'};
```

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

```
char s[5] = "Ciao";
```

- In quest'ultimo caso 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'  
...
```

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

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

Si utilizza lo specificatore di formato **%s**
NON bisogna usare il **&** per leggere la stringa

Stringhe: lettura e scrittura

- Analogamente per la scrittura:
elencando le singole componenti, così come ogni altro array :

```
...  
char str[4]; int i;  
for (i=0; str[i] != '\0'; i++)  
    printf("%c", str[i]);  
...
```

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

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

Si utilizza lo specificatore
di formato **%s**
Viene stampato tutto il contenuto
di *str* fino al terminatore **\0**

Stringhe: esempio 1

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*

Stringhe: esempio 1

Codifica: copia della stringa carattere per carattere

```
main() {  
    char s[] = "Oggi è una bella giornata";  
    char s2[40];  
    int i;
```

La dimensione deve essere tale da poter contenere la stringa

```
    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

Stringhe: esempio 1

Perché non fare così?

```
main() {  
    char s[] = "Oggi è una bella giornata";  
    char s2[40];  
  
    s2 = s;  
}
```

Errore di compilazione
Incompatible types in assignment

- Questo avviene perché le stringhe, essendo degli array, **non** possono essere manipolate nella loro interezza
- Vedremo (*più avanti nel corso, durante delle esercitazioni*) che esiste una libreria standard **<string.h>** contenente un insieme completo di funzioni per la manipolazione di stringhe

Stringhe: esempio 2

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*
- possiamo usare:
 - due boolean (**uguale e precede**)
 - tre boolean (**uguale, s1 precede s2, s2 precede s1**)
 - *un intero (negativo, zero, positivo)*

... scegliamo la terza via

Stringhe: esempio 2

Specifica dell'algoritmo:

- scandire uno a uno gli elementi *di equal 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

Stringhe: esempio 2

- **Codifica in C:**

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

L'esito del confronto è dato dal valore della variabile `stato`:

negativo → s1 precede s2

zero → s1 uguale s2

positivo → s2 precede s1