



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI  
INFORMATICA - SCIENZA E INGEGNERIA

# STRINGHE

COSIMO LANEVE

`cosimo.laneve@unibo.it`

CORSO 00819 – PROGRAMMAZIONE

# ARGOMENTI (SAVITCH, CAPITOLO 8)

1. array per il tipo `String`

2. la classe ~~standard~~ `string`

- \* più potente

- \* tratteremo **soltanto il primo modo** perchè l'altro richiede di conoscere le classi, quindi dovremmo aspettare la fine del corso

- \* se qualcuno lo conosce **non potrà usarlo**

- \* **non si potrà usare nelle prove di esame**

# ARRAY DI CHAR PER IL TIPO STRINGA

le **sequenze di caratteri** sono dette **stringhe**

le stringhe non sono un nuovo tipo di dato

- \* le stringhe sono **implementate** come **array di caratteri**
- \* in C++ la dichiarazione di una variabile di tipo stringa è:  
`char s[10] ;`
  - può memorizzare stringhe di al max 9 caratteri
  - l'ultimo carattere è il **carattere null** `'\0'` che indica la fine della stringa
  - il carattere null è **un carattere singolo**

# STRINGHE: ANCORA DETTAGLI

- \* un array di caratteri che non contiene `'\0'` **non è significativo**
- \* per contenere una stringa è **necessario sovrastimare** l'array
- \* fare attenzione a non uscire dalla dimensione dell'array con l'input
  - il **buffer overflow** è una tecnica di attacco hacker molto nota
- \* la funzione `sizeof(S)` ritorna la lunghezza dell'array `S`

# STRINGHE: DICHIARAZIONI E INIZIALIZZAZIONI

- \* per **dichiarare** una variabile di tipo stringa usare il pattern  

```
char Array_name[Maximum_String_Size + 1];
```

  - il +1 serve per il carattere '\0'
- \* per **inizializzare** una variabile di tipo stringa durante la dichiarazione 

```
char my_message[20] = "Hi there.";
```

  - '\0' aggiunto automaticamente
  - si può usare in alternativa 

```
char short_string[ ] = "abc"
```

la lunghezza è determinata automaticamente: in questo caso è 4

# STRINGHE: DETTAGLI

- \* dichiarare una stringa come `char s[10]` crea spazio per **solamente 9 caratteri**
  - il carattere `'\0'` è **necessario** e richiede una posizione
- \* ogni carattere dopo il null **non è significativo**

se il primo carattere della stringa è `'\0'` allora la stringa è vuota

- \* la presenza di `'\0'` **consente di determinare la lunghezza** della stringa
- \* **esempio:**

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
c	i	a	o	\0	a	b	c	d	e

# OPERAZIONI SU STRINGHE: STRLEN E STRCAT

\* `strlen(a_string)` ritorna il numero di caratteri nell'argomento

```
int x = strlen(a_string);
```

■ il carattere `'\0'` non conta

\* `strcat(a_string, b_string)` concatena due stringhe

■ il secondo argomento è giustapposto al primo

■ il risultato è memorizzato nel primo argomento

■ **esempio:**

`strcat` è poco sicura!

```
char string_var[20] = "The rain";  
strcat(string_var, "in Spain");
```

ora `string_var` contiene `"The rainin Spain"`

# LA FUNZIONE `STRNCAT`

- \* `strncat(a_string, b_string, n)` concatena il primo e i primi `n` caratteri del secondo argomento
- \* il terzo parametro specifica il numero di caratteri da concatenare (il secondo argomento viene troncato a quei caratteri)

## \* **esempio**

```
char a_string[30] = "The rain";  
strncat(a_string, " in Spain", 11);  
strncat(a_string, "and in Italy", 5);
```



```
a_string == "The rain in Spain"
```



```
a_string == "The rain in Spainand i"
```



# OPERAZIONI SU STRINGHE: `STRNCPY`

- \* il comando di assegnamento

```
a_string = "hello";
```

è **illegale** perchè il comando di assegnamento non opera correttamente con le stringhe

- \* il metodo standard è usare la funzione di libreria `strncpy`, definita in `cstring`:

```
#include <cstring>
. . .
char a_string[length];
strncpy (a_string, "Hello",size);
```

mette "Hello" seguito dal carattere '`\0`' in `a_string` se `size < length` altrimenti il '`\0`' non lo inserisce e il risultato non è una stringa!

- \* inserire sempre MANUALMENTE il '`\0`'

# STRNCPY E STRCPY

- \* `strncpy` usa un **terzo argomento** che rappresenta il numero massimo di caratteri da copiare

```
char destination[10];  
strncpy(destination, source, 9);
```

questo codice copia i primi 9 caratteri di `source` in `destination`, lasciando lo spazio per `'\0'`

**ATTENZIONE:** in questo caso lo `'\0'` è aggiunto implicitamente!

- \* alternativa meno sicura a `strncpy`: `strcpy`

- prende in input 2 stringhe
- **non verifica** la lunghezza della stringa primo argomento, quindi può tentare di scrivere **oltre la lunghezza dichiarata**

**ATTENZIONE:** lo `'\0'` NON è aggiunto da `strcpy` VA FATTO  
ESPLICITAMENTE!

# OPERAZIONI TRA STRINGHE: `strcmp`

- \* l'uguaglianza tra stringhe **non si esprime con "=="**
- \* si usa la funzione `strcmp` per confrontare variabili di tipo stringa
- \* **esempio:**

```
#include <cstring>
. . .
if (strcmp(c_string1, c_string2))
    cout << "Strings are not the same.";
else cout << "String are the same.";
```

**attenzione alla semantica di `strcmp` !**

# SEMANTICA DI `strcmp`

- \* "`strcmp`" confronta i codici numerici dei **caratteri corrispondenti** nelle due stringhe
- \* se le due stringhe sono le stesse allora `strcmp` ritorna 0 (**false**)
- \* al primo carattere differente
  - `strcmp` ritorna un **valore negativo** se il codice del carattere del primo parametro **è minore**
  - `strcmp` ritorna un **valore positivo** se il codice del carattere del primo parametro **è maggiore**
  - i valori non-zero sono interpretati come **vero**

# OUTPUT DI STRINGHE

\* le stringhe possono essere date in output

```
char news[20] = "stringhe,";  
cout << news << " Wow." << endl;
```

\* output:

```
> stringhe, Wow.
```

# INPUT DI STRINGHE

- \* è possibile fare input di stringhe con `"cin >>"`
- \* l'input **termina con uno spazio bianco**, il carattere `'\0'` viene aggiunto automaticamente
- \* **esempio:**

```
char a[80], b[80] ;  
cout << "Enter input: " << endl ;  
cin >> a >> b ;  
cout << a << b << "End of Output";
```

con input **"un buon pomeriggio"** stamperà

```
> Enter input:  
> unbuonEnd of Output
```

# LEGGERE UNA INTERA RIGA

- \* usare la funzione `cin.getline`
- \* `cin.getline` legge una intera riga inclusi gli spazi
- \* `cin.getline` ha **due** argomenti
  - il **primo** è la variabile di tipo stringa che riceverà l'input
  - il **secondo** è un intero (di solito la lunghezza dell'array passato come primo argomento) che determina il **massimo numero di caratteri** preso in input e che sarà memorizzato nella stringa
- \* **sintassi:** `cin.getline(A, m+1) ;`
  - ↑  
variabile di tipo stringa
  - ↑  
legge m caratteri  
un carattere serve per '\0'

# GETLINE: ESEMPIO E DETTAGLI

\* **esempio:**

```
char A[70] ;  
cin.getline(A, 10) ;  
cout << A ;
```

**video:**

```
> sei un grande  
> sei un gr
```



**ha memorizzato 9 caratteri + '\0'**

- \* `cin.getline` termina di leggere quando ha raggiunto il **numero di caratteri meno uno** specificato nel secondo argomento
- \* un carattere è riservato per il `'\0'`
- \* quindi `getline` può terminare la lettura anche se la linea non è stata letta completamente



# DALLE STRINGHE AI NUMERI

\* le funzioni di conversione

`atoi : string → int`

`atol : string → long int`

`atof : string → double`

si trovano nella libreria `cstdlib`

\* per usarle, aggiungere la direttiva

```
#include <cstdlib>
```

# DALLE STRINGHE AGLI INTERI

- \* quando si fa input di interi è **spesso conveniente** leggere l'input come una stringa e poi **convertire** la stringa in intero
  - perchè quando si legge una quantità in denaro c'è spesso "€"
  - quando si legge una percentuale c'è sempre il simbolo "%"
- \* per leggere un intero come sequenza di caratteri:
  - leggi l'input in una variabile di tipo stringa
  - rimuovere i caratteri indesiderati che si trovano in testa (non appena trova un carattere non cifra, `atoi` interrompe l'input)
  - usare la funzione `atoi` per convertire la stringa in intero
- \* **esempio:** `atoi("1234")` ritorna 1234  
`atoi("#123")` ritorna 0 perchè # non è una cifra

# DALLE STRINGHE AI LONGINT E DOUBLE

- \* le stringhe di cifre **più grandi** possono essere **convertiti** con la funzione `atol`
  - `atol` ritorna un `long int`
  - ci sono le stesse problematiche di `atoi` per i caratteri non cifra
- \* una stringa può essere convertita in tipo **double** con la funzione `atof`
  - `atof("9.99")` ritorna `9.99`
  - `atof("$9.99")` ritorna `0.0` perchè "\$" non è una cifra

# STRINGHE COME ARGOMENTI DI FUNZIONI

- \* quando le stringhe sono parametri formali o attuali di funzioni essi **sono considerati come array**
- \* se una funzione **cambia il valore** di un parametro stringa, allora è bene passare la lunghezza (per evitare accessi out-of-bound)
- \* se una funzione **non cambia il valore** di un parametro stringa, allora non è necessario passargli la lunghezza perchè essa è determinata da "`\0`"

# ESERCIZI

1. Scrivere una funzione che prende tre stringhe e stampa la più lunga
2. Scrivere una funzione che prende due nomi e stampa il primo dei due secondo l'ordine alfabetico
3. Definire una funzione

```
void parola(char str[], char c, char dest[])
```

che prende come parametri una stringa **str** e un carattere **c** e restituisce nel parametro **dest** la stringa corrispondente alla prima parola dentro **str** che inizia per **c** se presente. Si assuma che una parola sia una qualunque sequenza di caratteri diversi da spazio e che le parole siano separate tra loro da uno o più spazi.

4. Scrivere una funzione che prende in input una stringa **s** e la sua lunghezza e ritorna **true** se contiene le sottostringhe "gh" o "ch" oppure **false** in caso contrario. [esame 02/2018]

# LAVORARE CON LE STRINGHE A BASSO LIVELLO

- \* si può lavorare sulle stringhe usando le operazioni sugli array di caratteri e rispettando la convenzione su `'\0'`
- \* si elaborano i caratteri uno alla volta
- \* quando si passa una stringa a una funzione, solitamente non è necessario passarne la lunghezza
- \* la posizione del `'\0'` definisce dove la stringa termina
- \* se si estende la stringa occorre eliminare il `'\0'` e rimetterlo più avanti

# ESEMPI

- \* una funzione che restituisce la lunghezza di una stringa

```
int length(char A[]) {  
    int i = 0 ;  
    while (A[i] != '\0') i = i+1 ;  
    return(i) ;  
}
```

- \* una funzione che copia una sequenza di caratteri alfabetici in input in un array di **char**

```
void initialize_string(char A[], int l){  
    int i = 0 ;  
    char c ;  
    do {  
        cin >> c ;  
        if (((('a'<=c) && (c<='z')) || (('A'<=c) && (c<='Z')))) {  
            A[i] = c ;  
            i = i+1 ;  
        }  
    } while (i<l-1) ;  
    A[l-1] = '\0' ;  
}
```

# ATTENZIONE QUANDO SI LAVORA A BASSO LIVELLO

- \* se il carattere '\0' è perso il seguente codice produce un errore

```
int index = 0;
while (our_string[index] != '\0') {
    our_string[index] = 'X';
    index = index+1 ;
}
```

perchè ci potrebbe essere un **out-of-bound access**

- \* meglio il codice

```
int index = 0;
bool GOT_0 = false ;
while ((index < SIZE) && !GOT_0)
    if (our_string[index] != '\0') {
        our_string[index] = 'X';
        index = index+1 ;
    } else GOT_0 = true ;
```



# ESERCIZI

1. Scrivere una funzione che concatena due stringhe, mettendo il risultato nella prima
2. Scrivere una funzione che verifica se una stringa è composta da sole lettere maiuscole
3. Scrivere una funzione che elimina da una stringa tutti i caratteri dopo l'ennesimo

# STRINGHE: ERRORI COMUNI

- \* **non si può usare** questa alternativa per inizializzare:  

```
char short_string[ ] = {'a', 'b', 'c'};
```

  - perchè non aggiunge `'\0'`