UNIVERSITÀ
degli STUDI
di CATANIA

Array di caratteri in C. Funzioni di libreria

Corso di programmazione I (A-E / O-Z) AA 2025/26

Corso di Laurea Triennale in Informatica

Fabrizio Messina
fabrizio.messina@unict.it

Dipartimento di Matematica e Informatica

Rappresentazione delle stringhe in C

Una stringa è una sequenza di caratteri memorizzati in celle di memoria adiacenti (array).

```
1  char s1[] = {'m', 'y', '_', 's', 't', 'r', 'i', 'n', 'g', 0};
2  char s2[] = {'m', 'y', '_', 's', 't', 'r', 'i', 'n', 'g', '\0'};
3  char s3[] = {'m', 'y', '_', 's', 't', 'r', 'i', 'n', 'g'}; //NO!
```

Array di caratteri inizializzati mediante lista di inizializzatori: necessita il carattere finale *null terminator* '\0' anche detto carattere 'null' (codice ASCII 0);

La dichiarazione alla **linea 3 non va bene**: le funzioni (es: strlen()) che operano su array di caratteri non potranno identificare la fine della stringa.

Rappresentazione delle stringhe in C

```
1       char s[] = "my_string"; //OK
2
3       char *ps = "my_string"; // dovrebbe essere const..
```

Linea 1: inizializzatore finale carattere carattere 'null' aggiunto dal compilatore.

Un **letterale stringa** è un **puntatore** (costante!) ad array di caratteri: in quanto il compilatore memorizza il letterale in locazione di memoria (read-only) e conserva il puntatore.

Dunque istruzione alla linea 3 richiede conversione implicita da tipo `(const char *)` a tipo `(char *)`.

Rappresentazione delle stringhe in C

NB: **Accesso in scrittura ad elemento di ps darà errore a tempo di esecuzione..**

Rappresentazione delle stringhe in C

È buona norma inserire il qualificatore const..

```
1       const char *ps1 = "my_string"; //OK
2       ps1[3] = 'k'; //compilation error!
```

Qualsiasi tentativo di acceso in scrittura ad elemento dello array di caratteri (linea 2) darà **errore di compilazione**:

Il qualificatore `const` obbliga il programmatore a scrivere codice "safe".

Rappresentazione delle stringhe in C

Esempi svolti

`17_01.c`

`STR_01.c`

Funzioni della libreria standard per array di caratteri

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

Documentazione:

https://en.cppreference.com/w/c/string/byte

oppure..

https://devdocs.io/c-strings/

Tali funzioni possono prevedere diversi tipi di parametri formali:

- puntatori a caratteri (array di caratteri)
- letterali stringa (ovvero puntatori costanti)

Funzioni della libreria standard per array di caratteri

Copia di stringhe

```
char *strcpy (char * destination, const char * source);
```

```
char name[15];
strcpy(name, "pippo");
```

Copia la stringa source nell'area di memoria puntata da destination.

NB: **Nessun controllo sulla dimensione della area di memoria puntata dal parametro destination**: se stringa source più lunga, possibili problemi, anche di sicurezza (e.g. stack overflow).

$\rightarrow$ Meglio usare la funzione **strncpy()**.

Funzioni della libreria standard per array di caratteri

Confronto lessicografico.

```
int strcmp(const char *lhs, const char *rhs);
```

```
strcmp("pippo", "paperino"); // restituisce un int>0
strcmp("paperino", "pippo"); // restituisce un int<0
strcmp("pippo", "pippo"); // restituisce zero
```

Alla riga 1, dato che "pippo" $>$ "paperino", restituisce un int > 0.

Alla riga 2, dato che "paperino" $<$ "pippo", restituisce un int < 0.

Infine, restituisce zero in corrispondenza della riga 3 in quanto le due stringhe sono identiche.

Funzioni della libreria standard per array di caratteri

Ancora sul confronto lessicografico:

```
1    char s1[]="pippo";
2    char s2[]="paperino";
3    if(s1==s2){ //NO!
4       //...
5    }
```

Non usare operatore '==' per confrontare il contenuto di array di caratteri!

Funzioni della libreria standard per array di caratteri

Confronto lessicografico "length-bounded": imita il confronto ai primi count caratteri.

```
int strncmp(const char* lhs, const char* rhs, size_t count);
```

Copia di stringhe versione "length-bounded": copia i primi count caratteri di `src` nell'area puntata da `dest`.

```
char *strncpy(char *dest, const char *src, size_t count);
```

NB: Parametro formale `src` è generalmente dichiarato `const char *` (la funzione non deve modificare la stringa puntata da src).

Funzioni della libreria standard per array di caratteri

Ricerca di sottostringhe:

```
const char * strstr (const char *str1, const char *str2);
      char * strstr (      char *str1, const char *str2);
```

```
char s[] = "CesareGiulio";
char *found = strstr(s,"are");
printf("%s", found); // output: areGiulio
```

La funzione `strstr` restituisce un puntatore al primo carattere della prima occorrenza di str2 trovata in str1;

Funzioni della libreria standard per array di caratteri

Esempi svolti

`STR_02.c`

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

```c
#include <stdlib.h>
```

Conversione di stringhe a interi (int, long o long long):

```c
int       atoi( const char *str );
long      atol( const char *str );
long long atoll( const char *str );
```

Se stringa non rappresenta un numero atoi/atol/atoll potrebbe:

- **restituire la conversione dei primi caratteri delle stringa**, se questi rappresentano un numero. ES: per la stringa "22aabb" restituirebbe 22.

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

```
int       atoi( const char *str );
long      atol( const char *str );
long long atoll( const char *str );
```

- **restituire zero se la stringa non inizia con un numero**.
  NB: In questo modo **non è possibile distinguere** il caso "0" dagli altri casi..

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Conversione di stringhe a numeri in virgola mobile (float/double):

```
double atof( const char *str );
```

Il comportamento è identico a quello della funzione `atoi/atol/atoll`.

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Esempi svolti

17_02.c

STR_03.c

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Famiglia di funzioni strtoX (float/double):

```
#include <stdlib.h>
long strtol(const char *nptr, char **endptr, int base);

double strtod(const char *nptr, char **endptr);
float strtof(const char *nptr, char **endptr);
long double strtold(const char *nptr, char **endptr);
```

Alcune funzioni di conversione per long, double, float, long double (che dovrebbe essere 128 bit).

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Caratteristiche:

- restituisce il valore riconosciuto nella stringa rappresentata da nptr;
- pone nel secondo argomento (puntatore a puntatore), l'indirizzo di una cella che contiene il puntatore al **primo carattere non usato nella conversione**;

È possibile riconoscere il caso in cui nessuna conversione sia avvenuta, basta controllare che *endptr=nptr;

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Esempi svolti

`strtoX.c`

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Funzione `sscanf()` più "sofisticata": permette di estrarre piú di un elemento (non solo numeri) alla volta.

```
int sscanf(const char* buffer, const char* format,...);
```

Ricerca di elementi (stringhe e numeri) da estrarre da una stringa (contenuta nell'argomento `buffer`) sulla base di pattern definiti nel secondo argomento (`format`).

Salvataggio nelle aree di memoria specificate nella lista variabile di parametri dopo il secondo argomento.

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Esempio. **Lettura di un numero intero.**

```
1   int x;
2   const char *str = "1234";
3   sscanf(str, "%d", &x);
```

Comportamento analogo alla funzione atoi/l/ll(): se uno piu caratteri a inizio stringa sono validi la funzione restituisce la loro conversione.

ES: per "12aa34" `sscanf()` restituirà il numero 12.

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

Esempio. **Lettura di un numero in virgola mobile**

```
1    float y;
2    double x;
3    const char *str = "1234.56789012345";
4    sscanf(str, "%f", &y);
5    sscanf(str, "%lf", &x);
```

Lo specificatore '%f' significa `float`, mentre '%lf' significa `long float` ovvero `double`.

Conversioni numeriche: Array di caratteri $\longrightarrow$ Tipo numerico

È anche possibile specificare il numero di cifre del numero da leggere.

```
1    int y;
2    const char *str = "123456789";
3    sscanf(str, "%6d", &y);
4    printf("%d", y); //stampa 123456
```

Nel caso dei numeri in virgola mobile l'eventuale il carattere '.' va conteggiato nella lunghezza da specificare.

tipo numerico → stringa

`sprintf()`: "duale" rispetto alla `sscanf()`.

```
int sprintf(char* buffer, const char* format, ...);
```

Scrive nel buffer la stringa specificata dal parametro format con opportuni specificatori.

Le **variabili** numeriche (e non) sono specificate dal **terzo argomento** in poi.

tipo numerico → stringa

Esempio.

```
1    int a=10;
2    float x = 43.567;
3    double alpha = 123.456789123;
4    char str[100];
5    sprintf(str, " Intero a: %d, x: %.6f, alpha: %.12f", \
6      a, x, alpha);
7    printf("%s",str);
```

tipo numerico $\rightarrow$ stringa

Esempi svolti

`17_03.c`

`17_04.c`

Funzioni che operano su singoli caratteri

```
#include <ctype.h>
```

funzione	Valore di ritorno
`isalpha`	true se l'argomento è una lettera, false altrimenti
`isalnum`	true se l'argomento è lettera o numero, false altrimenti
`isdigit`	true se l'argomento è numero, false altrimenti
`isxdigit`	true se l'argomento è un carattere valido per la rappresentazione esadecimale: $0-9$, a-f, A-F

Funzioni che operano su singoli caratteri

```
#include <ctype.h>
```

funzione	Valore di ritorno
`isprint`	true se l'argomento è un carattere stampabile, false altrimenti
`ispunct`	true se l'argomento è un carattere di punteggiatura, false altrimenti
`islower`	true se l'argomento è minuscolo, false altrimenti
`isupper`	true se l'argomento è maiuscolo, false altrimenti
`isspace`	true se l'argomento è uno spazioo, false altrimenti

Funzioni che operano su singoli caratteri

```c
#include <ctype.h>
int toupper(int c);
int tolower(int c);
```

Entrambe le funzioni `toupper()` e `tolower()` prevedono un parametro formale int che rappresenta un carattere.

La funzione `toupper` (risp. `tolower`) restituisce la versione maiuscola (risp. minuscola) del carattere passato com argomento.

Funzioni che operano su singoli caratteri

Esempi svolti

`STR_04.c`

`STR_05.c`

Riferimenti e approfondimenti

[1] → Capitolo 8.

---

[1] Paul J. Deitel and Harvey M. Deitel.
C Fondamenti e tecniche di programmazione.
Pearson, 2022.

Homework H17.1

Definire un array di caratteri di lunghezza 15. Riempire tale array con caratteri random nel'intervallo [a-z].

Definire un array di caratteri mediante una lista di inizializzazione, che abbia almeno 5 caratteri. Stampare la stringa con una istruzione printf.

Definire un array mediante un inizializzatore che sia un letterale stringa. Stampare quindi la lunghezza della stringa, e la stringa stessa mediante una istruzione printf.

Definire un array di caratteri di sola lettura, mediante la definizione di un puntatore a carattere. Stampare quindi la stringa e la lunghezza della stringa stessa, mediante una istruzione printf.

Homework H17.2

Definire tre stringhe di lunghezza random che sia compresa tra 10 e 20.

Riempire le tre stringhe con caratteri pseudocasuali che appartengano agli intervalli [a-z, 0-9].

Eseguire un confronto lessicografico delle tre stringhe mediante la funzione `strcmp`.

Infine stampare le tre stringhe, una riga dopo l'altra dalla stringa minore alla stringa maggiore.

Homework H17.3

Scrivere un programma che chieda in input al'utente nome e cognome, da memorizzare in due array di caratteri distinti.

Definire quindi un array di caratteri che possoa contenere sia nome che cognome separati da un carattere ';', senza spazi.

Copiare nome e cognome mediante la funzione `strcpy` nel nuovo array di caratteri, senza dimenticare il carattere ';'.

Homework H17.4

Scrivere un programma che chieda all'utente di inserire alcuni verbi all'infinito presente (e.g. mangiare, spalare, dormire, etc), con un limite massimo sul numero di verbi che potrà inserire (es: 30).

Memorizzare i verbi in un array di stringhe di lunghezza prefissata (es: un array che potrà contenere 30 stringhe ovvero puntatori a caratteri).

Il programma dovrà stampare in ouput il numero di verbi con coniugazione "are", "ere", "ire".

Homework H17.5

Scrivere un programma che chieda all'utente di inserire tre numeri in virgola mobile. Il programma dovrà inizialmente memorizzare i tre numeri in altrettanti array di carateri.

Successivamente il programma dovrà memorizzare i tre numeri in altrettante variabili di tipo `double`:

- il primo numero mediante funzione di libreria `atof`;
- il secondo numero mediante funzione di libreria `sscanf`;
- il terzo numero mediante funzione di libreria `strtof`;

Il programma stamperà infine la media aritmetica dei tre numeri.

Homework

Homework H17.6 Scrivere un programma che chieda all'utente di inserire alcune stringhe, una alla volta, entro un limite massimo (es 30). Il programma memorizza le stringhe in appositi array di caratteri.

Ogni stringa in input potrà anche contenere spazi, ed il programma leggerà al massimo 20 caratteri per ogni stringa in input.

Il programma dovrà stampare, per ogni stringa memorizzata:

- la stringa stessa ;
- il numero di lettere minuscole presenti nella stringa;
- il numero di caratteri che rappresentano numeri, presenti nella stringa;
- il numero di spazi presenti nella stringa;