



UNIVERSITÀ
degli STUDI
di CATANIA

Oggetti string e array di caratteri in C++

Corso di programmazione I AA 2019/20

Corso di Laurea Triennale in Informatica

Prof. Giovanni Maria Farinella

Web: <http://www.dmi.unict.it/farinella>

Email: gfarinella@dm.unict.it

Dipartimento di Matematica e Informatica

Rappresentazione delle stringhe in C/C++

Una stringa è generalmente 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 che operano sulle stringhe avranno problemi.

Rappresentazione delle stringhe in C/C++

```
1    char s[] = "my_string"; // OK
2
3    //C: OK / C++11: Compilation Warning!
4    char *ps = "my_string";
```

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 memoria e conserva il puntatore.

Dunque istruzione alla linea 4 richiede conversione implicita da tipo (const char *) a tipo (char *).

Rappresentazione delle stringhe in C/C++

```
1    char s[] = "my_string"; // OK
2
3    //C: OK / C++11: Compilation Warning!
4    char *ps = "my_string";
```

Richiesta di conversione alla linea 4 **deprecato in C++11**:

- compilatore genera un warning;
- tentativo di accesso ad elemento array in scrittura darà errore a run-time!

Invece (vedi slide successiva)..

Rappresentazione delle stringhe in C/C++

```
1    const char *ps1 = " my_string"; //C: OK / C++11: OK  
2    ps1[3] = 'k'; //compilation error!
```

Linea 1 è OK:

Qualsiasi tentativo di accesso in scrittura ad elemento dello array di caratteri (linea 5) darà **errore di compilazione**:

Di fatto “obbliga” il compilatore a concepire scrivere codice corretto.

Esempi svolti

A20_01.cpp

STR_01.cpp

STR_02.cpp

STR_03.cpp

Funzioni di libreria per array di caratteri

```
#include <cstring>
```

Documentazione:

<https://en.cppreference.com/w/cpp/header/cstring>

Tali funzioni possono prevedere diversi tipi di parametri formali :

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

Funzioni di libreria per array di caratteri

Copia

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

```
char name[15]; ←  
strcpy(name, "pippo");  
cout << name; ↲
```

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).

Funzioni di libreria 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 di libreria 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!

Invece, è possibile gli operatori relazionali sugli oggetti `string`!

Funzioni di libreria 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 essere autorizzata a modificare la stringa puntata da src).

Funzioni di libreria 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(s2, "are");  
cout << found;      // output: areGiulio
```

La funzione `strstr` restituisce un puntatore al primo carattere della prima occorrenza trovata in `str2`. NB: la stringa restituita da `strstr` è una copia (in memoria) della occorrenza trovata.

NB: Dichiarazione di due prototipi, **coerente con restrizioni C++11** in merito a conversioni implicite di puntatori a costanti a puntatori.

Esempi svolti

STR_04.cpp

Conversioni numeriche: Array di caratteri → Tipo numerico

```
#include <cstdlib>
```

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

```
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 della stringa**, se questi rappresentano un numero. ES: per la stringa "22aabb" restituirebbe 22.

Conversioni numeriche: Array di caratteri → 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 → Tipo numerico

1.2

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 → Tipo numerico

Esempi svolti

A20_02.cpp

STR_05.cpp

Conversioni numeriche: Array di caratteri → Tipo numerico

Funzione `sscanf()` più “s sofisticata”: permette di estrarre più di un elemento (non solo numeri) alla volta.

Documentazione: <https://en.cppreference.com/w/cpp/io/c>

```
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 → 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/1/ll()`: se uno più 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 → 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 → 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  cout << y << endl; //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().

Documentazione: <https://en.cppreference.com/w/cpp/io/c>

```
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.

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

Parametro format quasi identico a format di sscanf(), ma:

- sscanf(): %10f indica lettura di 10 cifre compreso il punto.
- sprintf(): %.4f indica scrittura di un numero in virgola mobile con 4 cifre dopo la virgola.

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  cout << str;
```


Esempi svolti

A20_03.cpp A20_04.cpp STR_06.cpp

Altre funzioni che operano su caratteri

```
#include <cctype>
```

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
ishex	true se l'argomento è composto di cifre per rappresentazione esadecimale: 0 – 9, a-f, A-F

Altre funzioni che operano su caratteri

```
#include <cctype>
```

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 spazio, false altrimenti

Altre funzioni che operano su caratteri

```
#include <cctype>
int toupper(int c);
int tolower(int c);
```

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

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

Esempi svolti

STR_07.cpp STR_08.cpp

Oggetti string (C++)

```
1  #include <string>
2  string a = "my_string"; //C++ string object
3  //vs
4  char a[] = "my_string"; // C-string
```

Vantaggi derivanti dall'uso degli oggetti `string` rispetto agli array di caratteri:

- Numerose funzioni membro per la manipolazione di stringhe;
- **Overloading operatori** per costruire semplici espressioni (e.g. confronto) che in C richiedono invocazione funzioni (es: `strcmp(...)`);

Oggetti string (C++)

```
1  #include <string>
2  string a = "my_string"; //C++ string object
3  //vs
4  char a[] = "my_string"; // C-string
```

- maggiori controlli rispetto ai “bound” (lunghezza della aree di memoria sulle quali operare). ES: strcpy()!

Oggetti string (C++)

Classe string. Creazione di oggetti string.

```
1  string s1; //oggetto string "vuoto"  
2  string s2("pippo"); //valore iniziale  
3  string s3 = "test"; // operatore "=" per valore iniziale  
4  string s4(s3);
```

Linea 1: il costruttore non prende argomenti, stringa avente lunghezza zero.

Linea 2: inizializza con un letterale array di caratteri.

Linea 3: operatore di assegnamento per modificare il contenuto dell'oggetto string appena creato.

Linea 4: costruttore di copia 

Oggetti string (C++)

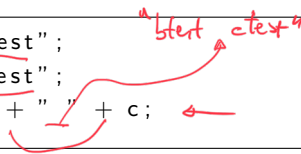
Operatore	Semantica
<u>=</u> $s_1 + s_2$	Assegnamento. Copia la stringa alla destra dell'operatore nell'oggetto string che sta alla sua sinistra.
+	Concatenazione. Concatenare due stringhe
+=	Aggiunge i caratteri della stringa che sta alla destra in coda alla stringa che sta a sinistra
>=, <= , ==, !=, < , >	Operatori relazionali per confronto lessicografico
<<, >>	Operatori di estrazione/inserimento per gli stream
[] $s_1[3]$	Operatore per singoli caratteri

Oggetti string (C++)

```
1 string b="btest";
```

```
2 string c="ctest";
```

```
3 string a = b + " " + c;
```



Linea 3: concatenazione stringhe a/o stringhe e letterali

Ma la seguente istruzione non è valida (operatore + applicato a letterali stringa).

```
cout << "btest" + " " + "ctest"
```

Oggetti string (C++)

Operatore [] vs metodo at().

```
1 string s = "0123456789"; //10 caratteri  
2 // nessun controllo sulla lunghezza della stringa!  
3 cout << s[10] << endl;  
4  
5 //at() solleva "eccezione" se indice >= s.length()  
6 cout << s.at(10) << endl;
```

NB: Metodo at() della classe string opera bound checking, ovvero controllo della lunghezza della stringa.

Oggetti string (C++)

Altri metodi della classe string

➡ https://en.cppreference.com/w/cpp/string/basic_string

- string vs array di caratteri (C-string): `c_str`, `data`;
- Modifica/manipolazione: append, `push_back`, `assign`, `clear`, copy, `erase`, `insert`, `replace`, `swap`;
- Gestione dello spazio in memoria: `capacity`, `empty`, `length`, `resize`, `size`
- Ricerca/estrazione di sottostringhe: `find`, `substr`
- confronto: `compare`

Esempi svolti

A20_06.cpp

STR_02.cpp

STR_03.cpp

STR_09.cpp

STR_10.cpp

STR_11.cpp

FINE