

Corso “Programmazione 1”

Capitolo 07: Stringhe e File di Testo

Docente: **Marco Roveri** - marco.roveri@unitn.it
Esercitori: **Giovanna Varni** - giovanna.varni@unitn.it
Andrea E. Naimoli - andrea.naimoli@unitn.it
Giacomo Colle - giacomo.colle@unitn.it
C.D.L.: Informatica (INF)
A.A.: 2025-2026
Luogo: DISI, Università di Trento
URL: <https://shorturl.at/cxAAS>



Ultimo aggiornamento: 21 ottobre 2025

Terms of Use and Copyright

USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2025-2026.

SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

- 1 Stringhe
- 2 Argomenti da linea di comando
- 3 I/O con File di Testo

Stringhe (C)

- Una stringa C è un array di **char**, il cui ultimo elemento è il carattere nullo (' \0 ')
 - Esempio: **char** stringa[] = "Ciao";
 - Equivalente a **char** stringa[] = { 'C', 'i', 'a', 'o', '\0' };
⇒ contiene 5 elementi: i 4 caratteri della costante stringa e il carattere nullo che viene inserito automaticamente
- ⇒ La dimensione dell'array deve essere maggiore di almeno 1 rispetto al numero di caratteri che si vuole rappresentare

Nota:

dato che negli array non è definito l'assegnamento, l'uso di una costante stringa per specificare il valore di una stringa è permesso **solo nell'inizializzazione**

- L'esempio di cui sopra:
{ STRINGS/strings2.cc }

Input e Output di Stringhe

- Gli operatori di I/O `>>`, `<<` operano direttamente su stringhe!
- L' **operatore di ingresso** `>>`:
 1. legge caratteri da `cin`
 2. li memorizza in sequenza finché non incontra una spaziatura (che non viene letta)
 3. memorizza `'\0'` nella stringa dopo l'ultimo carattere letto
 4. termina l'operazione
- L' **operatore di uscita** `<<` scrive in sequenza su `cout` i caratteri della stringa, fino al primo `'\0'` (che non viene scritto)

Esempio

Legge una stringa di al più 255 caratteri e la stampa:

```
char buffer[256];  
cin >> buffer;  
cout << buffer;
```

Esempi

- input/output di stringhe, usare `./a.out < inputs/in:`
{ STRINGS/strings3.cc }
- variante, con cin-loop:
{ STRINGS/strings3_cinloop.cc }

Alcune funzioni utili della libreria `<iostream>` (Ripasso)

- `cin.eof()`: ritorna un valore diverso da 0 se lo stream `cin` ha raggiunto la sua fine (End Of File)
 - va usato sempre dopo almeno un'operazione di lettura
 - richiede un separatore dopo l'ultimo elemento letto
- `cin.fail()`: ritorna un valore diverso da 0 se lo stream `cin` ha rilevato uno stato di errore (e.g. stringa per `int`) o un end-of-file
 - va usato dopo almeno un'operazione di lettura
 - non richiede un separatore dopo l'ultimo elemento letto
- `cin.clear()`: ripristina lo stato normale dello stream `cin` dallo stato di errore
- uso di `cin.eof()` – usare `inputs/in1` e `inputs/in1.bis`:
{ STRINGS/converti.cc }
- uso di `cin.fail()` – usare `inputs/in1`, `inputs/in1.bis` e `inputs/in1.tris`:
{ STRINGS/converti1.cc }
- uso di `cin.clear()` – usare `inputs/in1.tris`:
{ STRINGS/converti2.cc }

Alcune funzioni utili della libreria `<iostream>` II

Nelle funzioni seguenti `s` è una stringa, `c` è un carattere e `n` un intero:

- `cin.getline(s, n)`: legge da `cin` una riga in `s` fino a capo linea, per un massimo di `n-1` caratteri (lo `'\n'` non viene letto)
 - restituisce (un oggetto equivalente a) 0 se incontra `eof`
- `cin.get(c)`: legge da `cin` in `c` un singolo carattere (spaziature comprese), restituisce `c` (`'\0'` se `c` è `eof`)
- `cout.put(c)`: scrive su `cout` il singolo carattere `c`
- uso di `cin.getline(char *, int)`:
{ STRINGS/strings4_while.cc }
- uso di `cin.get` (usare inputs/silvia.txt come input):
{ STRINGS/agosti.cc }
- Esempio uso di `cin.put`:
{ STRINGS/strings7.cc }

Alcune funzioni utili della libreria `<cstring>`

Nelle funzioni che seguono `s` e `t` sono stringhe e `c` è un carattere:

- `strlen(s)`: restituisce la lunghezza di `s` escluso `'\0'`
- `strchr(s, c)`: restituisce un puntatore alla prima occorrenza di `c` in `s`, oppure **NULL** se `c` non si trova in `s`
- `strrchr(s, c)`: come sopra ma per l'ultima occorrenza di `c` in `s`
- `strstr(s, t)`: restituisce un puntatore alla prima occorrenza della sottostringa `t` in `s`, oppure **NULL** se `t` non si trova in `s`
- `strcpy(s, t)`: copia `t` in `s` e restituisce `s`
- `strncpy(s, t, n)`: copia `n` caratteri di `t` in `s` e restituisce `s`. **Se non c'è lo `'\0'` negli `n` caratteri la stringa `s` non è ben formata!**
- `strcat(s, t)`: concatena `t` al termine di `s` e restituisce `s`
- `strncat(s, t, n)`: concatena `n` caratteri di `t` al termine di `s` e restituisce `s`. **La stringa di destinazione contiene sempre `'\0'`, vengono copiati `n` caratteri e sempre aggiunto `'0'`.**
- `strcmp(s, t)`: restituisce un valore negativo, nullo o positivo se `s` è alfabeticamente minore, uguale o maggiore di `t`
- **Prestare molta attenzione nelle funzioni sopra allo `'\0'` e che dimensioni delle stringhe di destinazioni siano in grado di contenere risultato!**

Esempi

- `strlen:`
`{ STRINGS/strings13.cc }`
- `strchr, strrchr, strstr` (ricerca di caratteri e stringhe in una stringa):
`{ STRINGS/strings14.cc }`
- `strcpy:`
`{ STRINGS/strings15.cc }`
- `strncpy:`
`{ STRINGS/strings16.cc }`

Esempi II

- `strcat` (attenzione, c'è un errore, dove?):
{ `STRINGS/strings17.cc` }
- versione corretta:
{ `STRINGS/strings17_correct.cc` }
- (compilare con `-fno-stack-protector`) effetto catastrofico:
{ `STRINGS/strings17_catastrophic.cc` }
- `strncat` (può dare errore a seconda di opzioni di compilazione):
{ `STRINGS/strings18.cc` }
- `strcmp`:
{ `STRINGS/strings19.cc` }

Esercizi proposti

Esercizi proposti:

{ STRINGS/ESERCIZI_PROPOSTI.txt }

Argomenti da linea di comando

- In C++ è possibile passare ai programmi argomenti (es. valori numerici, nomi di file,...) **direttamente da linea di comando**

```
> ./a.out 1000 22.5 miofile
```

- Possibile tramite due **parametri formali predefiniti** della funzione `main`:

```
int main (int argc, char * argv[]) {}
```

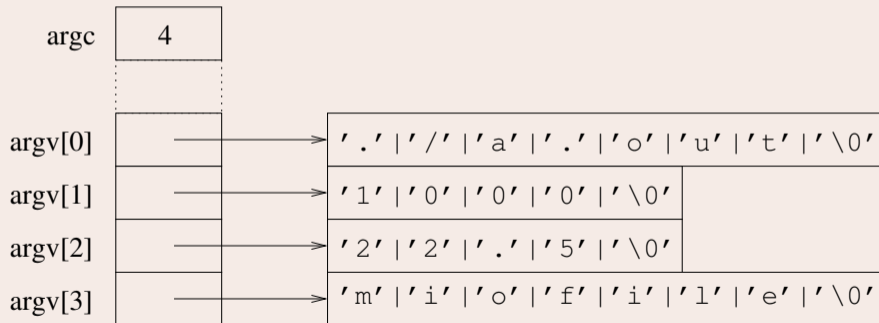
- l'intero `argc`, in cui viene automaticamente copiato il numero delle parole della riga di comando ("`./a.out`" o analogo inclusa)
- l'array di puntatori a caratteri (stringhe) `argv` in cui vengono automaticamente copiate le parole della linea di comando

Nota

Gli argomenti sono **stringhe**: se rappresentano numeri, devono essere convertiti tramite le funzioni `atoi` o `atof` della libreria `<cstdlib>`

Argomenti da linea di comando II

```
int main (int argc, char * argv[])  
{...}  
> ./a.out 1000 22.5 miofile
```



Esempi

- Esempio generico di uso di argc&argv:
{ STRINGS/argcargv.cc }
- .. con parametri numerici:
{ STRINGS/iva1.cc }

Stream e I/O su File di Testo

- In C++ sono possibili operazioni di I/O **direttamente da file di testo** (senza usare `<`, `>`) tramite la libreria `<fstream>`
- È possibile **definire stream**, a cui associare (i nomi di) file di testo.
- Lo stream viene **aperto** e associato al nome di un file tramite il comando **open**, in tre possibili modalità
 - **lettura** da file,
 - **scrittura** su file,
 - **scrittura a fine file (append)**.
- Lo stream può essere utilizzato per tutte le operazioni di lettura [resp. scrittura] a seconda della modalità di apertura
- Uno stream, quando è stato utilizzato, può essere chiuso mediante la funzione **close**

`fstream` “eredita” da `iostream` sostanzialmente tutti i suoi operatori e funzioni di lettura e scrittura, nelle rispettive modalità
(Es: `<<`, `>>`, `get`, `put`, `getline`, `eof`, `fail`, `clear`,...)

Operazioni su Stream: Sintassi

- **Definizione** di uno stream:

- Sintassi: `fstream nomestream;`
- Esempio: `fstream myin, myout, myapp;`

- **Apertura** di uno stream:

- Sintassi: `nomestream.open(nomefile, modo);`
- Es:

```
myin.open("ingresso.txt", ios::in); //lettura  
myout.open("uscita.txt", ios::out); //scrittura  
myapp.open("uscita2.txt", ios::out|ios::app); //app.
```

- **Utilizzo** di uno stream:

- Sintassi: analoga a `cin` e `cout`
- Es:

```
myin >> a; myout << x; myin.get(c); myapp.put(c);...
```

- **Chiusura** di uno stream:

- Sintassi: `nomestream.close();`
- Es: `myin.close(); myout.close();`

Apertura di un file

- Apertura in modalità **lettura** (`ios::in`):
 - il file associato deve già essere presente
 - il puntatore si sposta all'inizio dello stream
- Apertura in modalità **scrittura** (`ios::out`):
 - il file associato se non è presente viene creato
 - il puntatore si posiziona all'inizio dello stream (sovrascrivendo il file)
- Apertura in modalità **append** (`ios::out | ios::app`):
 - il file associato se non è presente viene creato
 - il puntatore si posiziona alla fine dello stream

Chiusura di uno stream

- Alla fine del programma tutti gli stream aperti vengono automaticamente chiusi
- Una volta chiuso,
 - uno stream può essere riaperto in qualunque modalità e associato a qualunque file
 - uno stream per essere utilizzato deve essere riaperto in qualunque modalità e associato a qualunque file

Nota:

È buona prassi di programmazione **chiudere** ogni stream aperto quando non più necessario (il sistema operativo ha un limite sul numero di file che possono essere aperti per un programma, vedi `ulimit -a` su bash).

Uso di `fstream` con `argc` e `argv`

- È desiderabile poter passare i nomi dei file al programma:

- Es: `> ./a.out pippo pluto`

⇒ nomi dei file passati tramite `argc` e `argv`:

```
int main (int argc, char * argv[])
```

```
    fstream myin,myout;
```

```
    myin.open(argv[1],ios::in);
```

```
    myout.open(argv[2],ios::out);
```

- È necessario gestire l'errore utente e la mancata apertura:

```
if (argc!=3) {
```

```
    cerr << "Usage: ./a.out_<source>_<target>\n";
```

```
    exit(0);
```

```
}
```

```
if (myin.fail()) {
```

```
    cerr << "Il_file_" << argv[1] << "_non_esiste\n";
```

```
    exit(0);
```

```
}
```

Esempi

- esempio di uso di `fstream` (usa `inputs/in1` e `inputs/in1.bis`):
{ IO_SU_FILES/converti1.cc }
- come sopra, con `cin` loop:
{ IO_SU_FILES/converti2.cc }
- effettua una copia di un file :
{ IO_SU_FILES/copiafile.cc }
- appende un file ad un altro file:
{ IO_SU_FILES/appendifile.cc }

Esercizi proposti

Esercizi proposti:

{ IO_SU_FILES/ESERCIZI_PROPOSTI.txt }