

# Algoritmi (modulo di laboratorio)

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano



- Lezioni: Martedì 8.30 - 10.30 in aula 8    **Martedì 15.30 - 18.30 in aula 2**  
**Mercoledì 10.30 - 13.30 in aula 2**    Venerdì 10.30 - 12.30 in aula 4
- Ricevimento: **su appuntamento** (Dipartimento di Informatica)
- E-mail: **roberto.cordone@unimi.it**
- Pagina web: **<http://homes.di.unimi.it/~cordone/courses/2025-algo/2025-algo.html>**
- Sito Ariel: **<https://myariel.unimi.it/course/view?id=3928>**

Lezione 2: Stringhe, stream, parsing, parametri del main Milano, A.A. 2024/25

# Stringhe di caratteri

In C **stringa** è

- qualsiasi sequenza di caratteri
- terminata dal carattere `'\0'` (**terminatore**)  
che non è significativo

Per rappresentarla esplicitamente, basta racchiuderla fra virgolette

Esempio: "Questa e' una stringa."

*(il terminatore segue l'ultimo carattere ed è implicito)*

Le variabili che conservano stringhe si dichiarano come vettori di caratteri

```
char s[ROW_LENGTH];
```

con lo spazio sufficiente a contenere anche il terminatore

Il primo terminatore contenuto nel vettore tronca la stringa

'p'	'r'	'o'	'\0'	'v'	'a'	'\0'
0	1	2	3	4	5	6

 vale "pro"

Non c'è controllo che un vettore di caratteri contenga un terminatore

*(questo può causare errori)*

# Lunghezza di una stringa e copia

La **libreria `string.h`** fornisce **funzioni per gestire stringhe di caratteri**  
Per poterle usare bisogna includere la libreria: `#include <string.h>`

La funzione

`strlen(s)`

fornisce la lunghezza di una stringa

**Non si possono copiare stringhe con l'operatore di assegnamento (=)**

La funzione

`strcpy(dest,orig)`

copia la stringa *orig* nella stringa *dest*

- **il C non controlla che la stringa *dest* possa contenere *orig*:**  
se *dest* è più corta, la copia eccede i limiti e sporca altri dati

La funzione `strncpy(dest,orig,n)` copia al max. i primi *n* caratteri

Non si possono confrontare stringhe con l'operatore ==

`strcmp(s1,s2)`

confronta le stringhe *s1* e *s2* in modo lessicografico (dizionario)

- 1 scorre in parallelo le due stringhe per  $i \geq 0$  caratteri fino a trovare
  - due caratteri diversi ( $s1[i] \neq s2[i]$ )
  - oppure il termine di una delle due stringhe
- 2 restituisce
  - valore nullo se entrambe le stringhe sono terminate
  - valore negativo se *s1* termina o  $s1[i] < s2[i]$
  - valore positivo se *s2* termina o  $s1[i] > s2[i]$

Si dice *stream* qualsiasi sorgente di dati in ingresso e qualsiasi destinazione per i risultati in uscita

- tastiera
- video
- file su disco, CD, DVD, memorie flash
- dispositivi di comunicazione (porte di rete, stampanti, ecc. . . )

La libreria `stdio.h` tratta tutti gli *stream* allo stesso modo (per quanto possibile)

- rappresentandoli con puntatori a file (`FILE *`)
- su cui operano funzioni simili o identiche

# Gli *stream standard*

Esistono tre *stream standard*, che non occorre definire, aprire e chiudere

- lo *standard input* (`stdin`), ovvero la *tastiera*
- lo *standard output* (`stdout`), ovvero il *video*
- lo *standard error* (`stderr`), ovvero il *video*

Quando si chiama un programma, il sistema operativo può *reindirizzare gli stream standard*, cioè *modificarne il significato*

- *programma < nomefile* indica che *si ricevono i dati dal file nomefile anziché da tastiera* (`stdin` punta il file *nomefile*)
- *programma > nomefile* indica che *si stampano i risultati sul file nomefile anziché a video* (`stdout` punta il file *nomefile*)
- *programma 2> nomefile* indica che *si stampano i messaggi di errore sul file nomefile anziché a video* (`stderr` punta il file *nomefile*)

**Parsing** (**analisi sintattica**) è il **processo che analizza una data sequenza di simboli per determinarne la struttura grammaticale**

Questa operazione è fondamentale perché serve ad acquisire i dati del programma da uno **stream in ingresso**

- dati digitati su tastiera
- stringhe di caratteri
- file su disco
- altre fonti esterne

**Parser** è un **programma che esegue tale analisi non banale**

# La funzione scanf

La principale funzione per il parsing di testo fornito da tastiera è

```
int scanf(char *formato, ...)
```

- ha un **numero variabile di parametri** ( $p \geq 1$ )
  - la **stringa di formato** definisce il *pattern* che si cerca di riconoscere nello stream in ingresso
  - $p - 1$  **puntatori** sono gli indirizzi delle celle dove conservare i valori riconosciuti nello stream in ingresso
- **termina alla fine della stringa o al primo oggetto non riconosciuto**
- restituisce il **numero di oggetti del *pattern* ritrovati nello stream**

Il resto dello stream resta disponibile per la chiamata seguente



# La stringa di formato

Stringa di formato e stream in ingresso sono compatibili quando

- ogni carattere non separatore nella stringa, cioè diverso da spazio, a capo e tabulazione, corrisponde allo stesso carattere nello stream
- uno o più separatori consecutivi nella stringa corrispondono a zero o più separatori consecutivi nello stream
- le specifiche di conversione nella stringa (`%. . .`) corrispondono a zero o più separatori consecutivi e una sequenza di caratteri che descrive un oggetto del tipo indicato dalla specifica (numero intero, numero reale, singolo carattere, parola, . . .)

Esempio:

La stringa di formato "Numero %d/%d" corrisponde allo stream "Numero" seguito da zero o più spazi e da due interi separati da '/'

Quindi è compatibile con

"Numero3/5", "Numero 15/ 2", "Numero -6/ 0", ecc. . .

Non è compatibile con "Numero 10 /5"

Indica il tipo di oggetto atteso nella posizione corrente dell'ingresso

%d	int decimale
%u	unsigned int decimale
%f	float in notazione decimale
%e	float in notazione scientifica
%c	char
%s	stringa priva di separatori
%[set]	stringa fatta con i caratteri elencati in set

Ogni puntatore indica dove copiare l'oggetto corrispondente

L'istruzione `scanf("%d/%d/%d", &giorno, &mese, &anno);`

- cerca tre numeri interi separati da '/'
- li assegna alle variabili `giorno`, `mese` e `anno`
- restituisce il numero di interi trovati

Ingresso	Istruzione	n	i	j
12 , 34	<code>n = scanf("%d%d",&amp;i,&amp;j);</code>	1	12	invariato
12 , 34	<code>n = scanf("%d,%d",&amp;i,&amp;j);</code>	1	12	invariato
12 , 34	<code>n = scanf("%d ,%d",&amp;i,&amp;j);</code>	2	12	34
12 , 34	<code>n = scanf("%d, %d",&amp;i,&amp;j);</code>	1	12	invariato
12 , 34	<code>n = scanf("%d , %d",&amp;i,&amp;j);</code>	2	12	34

Suggerimento: si noti la posizione della virgola nella stringa di formato

I **modificatori di lunghezza** alterano il tipo dell'oggetto

<code>%hd</code>	short decimale
<code>%hu</code>	unsigned short decimale
<code>%ld</code>	long decimale
<code>%lu</code>	unsigned long decimale
<code>%lf</code>	double
<code>%Lf</code>	long double

Un numero (**larghezza massima**) indica che l'oggetto trovato in ingresso deve contenere un massimo numero dato di caratteri: con ingresso "2010", l'istruzione `scanf("%2d",&i);` assegna a `i` il valore 20

Un asterisco (**soppressore**) indica che l'oggetto trovato in ingresso non va assegnato ad alcun puntatore: `scanf("%*d",&i);` non altera `i`

Ingresso	Istruzione	Risultati
12 34	<code>n = scanf("%*d%d", &amp;i);</code>	n = 1 i = 34
Un due tre	<code>n = scanf("%*s%s", s);</code>	n = 1 s = "due"
12345	<code>n = scanf("%1d%2d%3d", &amp;i, &amp;j, &amp;k);</code>	n = 3 i = 1 j = 23 k = 45
123456	<code>n = scanf("%2d%2s%2d", &amp;i, s, &amp;j);</code>	n = 3 i = 12 s = "34" j = 56

Si noti la mancanza della & prima di s

```
int printf(char *formato, ...)
```

è la principale funzione per la stampa a video

Funziona esattamente come la funzione `scanf`

Ha un numero variabile di parametri  $p \geq 1$

- la stringa di formato definisce un *pattern* che si ricostruisce in uscita
- $p - 1$  oggetti dai quali la funzione trae i risultati da stampare

Restituisce il numero di oggetti del *pattern* stampati in uscita

# Parsing di stringhe

Anche le stringhe di caratteri si possono vedere come *stream* e quindi si possono gestire con funzioni simili

```
int sscanf(char *stringa, char *formato, ...)
```

- riconosce nella *stringa* (primo parametro)
- gli elementi forniti dalla stringa di *formato* (secondo parametro)
- assegna gli oggetti riconosciuti ai puntatori (parametri successivi)

Esempio:

```
char s[10+1];  
int giorno, mese, anno;  
  
strcpy(s, "14/05/2010");  
sscanf(s, "%d/%d/%d", &giorno, &mese, &anno);
```

**sprintf genera stringhe formattate come printf stampa a video**

# Apertura di un file di testo (1)

I **file di testo** sono costituiti da **sequenze di caratteri organizzati in righe**, separate dal carattere speciale `'\n'`

Per usare un file occorre aprirlo con il comando

```
FILE *fopen(char *nomefile, char *modo)
```

che specifica

- il nome del file da aprire e la posizione su disco (*path*)
- il modo in cui usarlo
  - `"r"`: in **lettura**, ponendosi **al principio** del file
  - `"w"`: in **scrittura**, ponendosi **al principio** del file
  - `"a"`: in **accodamento**, ponendosi **alla fine** del file

Il *path* può essere assoluto o relativo (al file eseguibile o al progetto)



# Apertura di un file (2)

La funzione `fopen` restituisce un puntatore al file per poterlo usare

- se il file non esiste
  - in lettura, restituisce `NULL`
  - in scrittura e accodamento, ne crea uno vuoto
- se il file non può essere aperto o creato
  - restituisce `NULL`

Aperto un file, la posizione accessibile è

- il principio del file se si è aperto il file in lettura o scrittura
- la fine del file se si è aperto il file in accodamento

Dopo l'uso, il file va chiuso con l'istruzione

```
int fclose(FILE *stream)
```

Tutti gli stream di ingresso sono gestiti allo stesso modo

```
int fscanf(FILE *stream, char *formato, ...)
```

funziona come `scanf`, salvo richiedere il puntatore allo stream usato:

- interpreta il contenuto dello *stream*
- in base alla stringa di formato
- assegna gli oggetti riconosciuti ai puntatori che seguono
- restituisce il numero di oggetti assegnati

Esempio:

```
FILE *fp;  
int giorno, mese, anno;  
fp = fopen("prova.txt", "r");  
fscanf(fp, "%d/%d/%d", &giorno, &mese, &anno);
```

# Terminazione di un file

## Se si arriva al termine di un file

- la funzione `fscanf` restituisce il numero di oggetti assegnati
- se non ne ha assegnati, restituisce la costante simbolica `EOF`

```
int fscanf(FILE *stream, char *formato, ...)
```

N.B.: `fscanf` restituisce `EOF` se si trova esattamente al termine, non se il file termina durante il *parsing*

## Dopo il fallimento di un'operazione di lettura

- la funzione `feof` restituisce vero, cioè un valore intero non nullo

```
int feof(FILE *stream)
```

# Lettura di righe

Si può leggere un'intera riga da tastiera con l'istruzione

```
char *gets(char *s)
```

- opera sullo *stream* `stdin`
- legge una riga compreso il carattere terminale `'\n'`
- restituisce la stringa `s` escluso `'\n'`; se fallisce, restituisce `NULL`

Si può leggere un'intera riga da file con l'istruzione

```
char *fgets(char *s, int n, FILE *stream)
```

- opera sullo stream specificato
- legge una riga compreso il carattere terminale `'\n'`, ma legge al massimo `n` caratteri e non include `'\n'`
- restituisce la stringa `s` compreso il carattere terminale `'\n'`
- restituisce la stringa `s`; se fallisce, restituisce `NULL`

*È consigliabile sostituire `gets(s)` con `fgets(s,n,stdin)`*

```
int fprintf(FILE *stream, char *formato, ...)
```

funziona esattamente come printf e sprintf:

- scrive sullo *stream*
- nel *formato* specificato dalla relativa stringa
- il valore degli oggetti che seguono

```
int *fputs(char *s, FILE *stream)
```

scrive la stringa *s* sullo *stream* di uscita senza aggiungere '\n'

```
int *puts(char *s)
```

scrive la stringa *s* sullo stdout con un '\n' aggiuntivo

Restituiscono EOF se falliscono, un valore non negativo altrimenti

# Il main come funzione

Anche il `main` è una funzione, chiamata dal sistema operativo

- ha dei `dati`, descritti dai parametri formali nell'intestazione
- ha un `risultato`, descritto dal tipo restituito nell'intestazione

Tuttavia

- il numero e tipo dei parametri non è noto a priori
- il risultato è sempre intero

Questo apre una serie di domande

- come si definisce il numero e tipo dei parametri?
- come si passano i parametri attuali?
- a che serve e come si recupera il risultato?

# Parametri formali e attuali

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

I parametri formali non sono definiti direttamente, bensì attraverso

- `int argc`, che è il numero dei parametri
- `char *argv[]`, che è un vettore dinamico di lunghezza `argc` composto da stringhe che descrivono i parametri

I parametri attuali vengono passati attraverso la linea di comando

- `argv[0]` è il nome del programma
- `argv[1]` è il primo parametro
- `argv[2]` è il secondo parametro
- ...

Il risultato di un programma C è sempre un intero

La libreria `stdlib.h` definisce le costanti simboliche

- `EXIT_SUCCESS` (0) da usare se il programma ha avuto successo
- `EXIT_FAILURE` (-1) da usare in caso di errore

Si usa specificare il tipo di errore definendo altre costanti simboliche

Il valore numerico viene restituito al sistema operativo

- alcuni ambienti di compilazione lo indicano all'utente
- i file batch o script lo possono usare