



UNIVERSITÀ  
degli STUDI  
di CATANIA

DIPARTIMENTO DI  
MATEMATICA E INFORMATICA

# Elementi base di C++

Alessandro Ortis

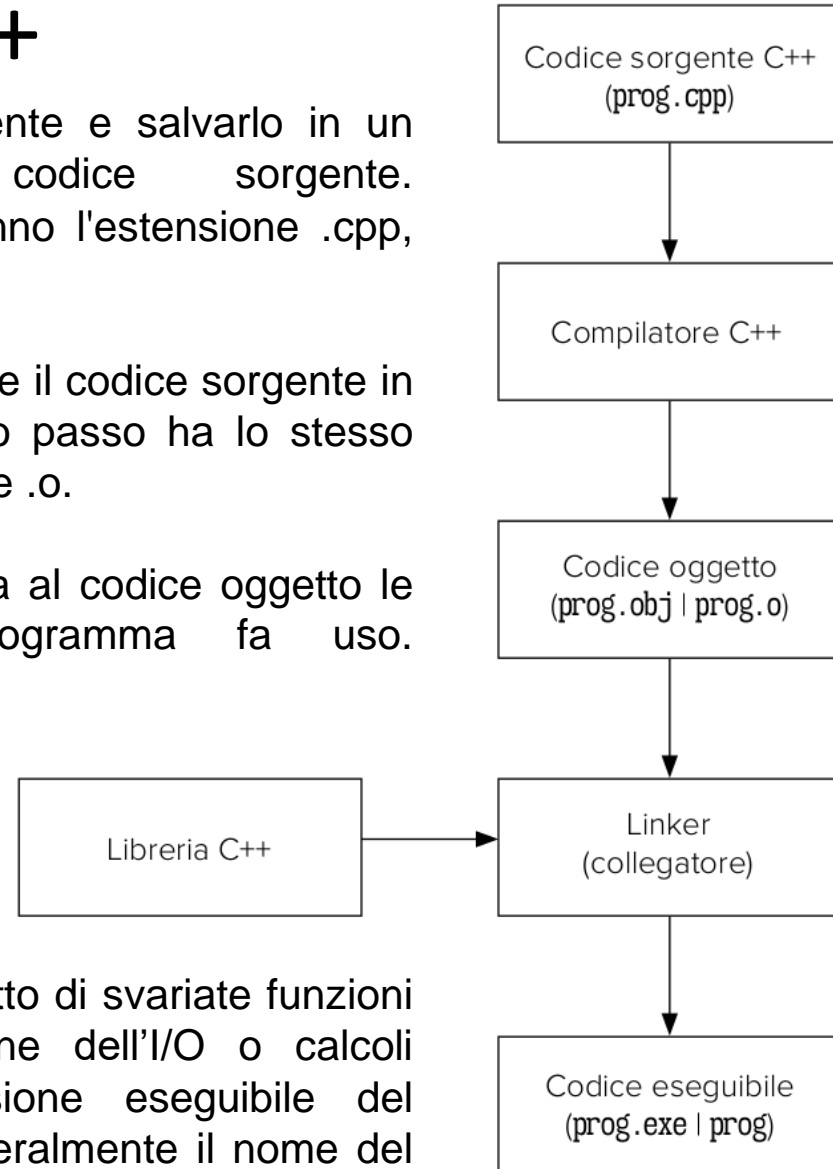
Image Processing Lab - [iplab.dmi.unict.it](http://iplab.dmi.unict.it)

[www.dmi.unict.it/ortis/](http://www.dmi.unict.it/ortis/)



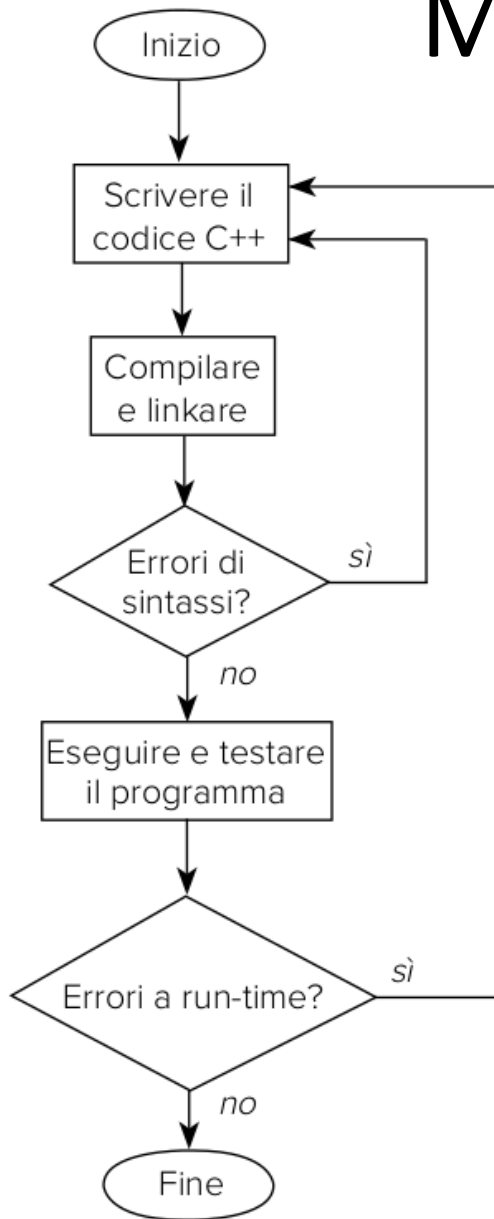
# Costruzione di un programma in C++

- 1) editor di testo per scrivere il programma sorgente e salvarlo in un file chiamato file sorgente o codice sorgente. Per convenzione, i programmi scritti in C++ hanno l'estensione .cpp, mentre quelli scritti in C hanno l'estensione .c.
- 2) Compilare il codice sorgente. Il compilatore traduce il codice sorgente in codice oggetto. In C++, il file prodotto in questo passo ha lo stesso nome di quello sorgente ma estensione .obj oppure .o.
- 3) Se il passo 2 ha avuto successo, il linker collega al codice oggetto le funzioni di libreria C++ di cui il programma fa uso.



La libreria standard del C++ contiene il codice oggetto di svariate funzioni che realizzano compiti comuni, come la gestione dell'I/O o calcoli matematici. Il collegamento produce una versione eseguibile del programma che viene posta in un file che ha generalmente il nome del sorgente ed estensione .exe sotto Windows e nulla sotto Linux.

# Messa a punto di un programma in C++



- 1) Scrittura del codice sorgente sul computer (salvare con un nome; ad esempio, demo.cpp)
- 2) Compilazione (compiler) del codice sorgente (demo.obj)
- 3) Collegare (linker) il codice oggetto prodotto con quello delle librerie dichiarate nel programma sorgente (normalmente questo passo è svolto insieme a quello precedente)
- 4) Il compilatore/collegatore individua eventuali errori di sintassi che devono essere corretti e quindi si torna al punto 1
- 5) Collaudo del programma. Se il programma non fa quello che il programmatore si aspettava si torna al punto 1.

```
#include <iostream>  ← File di libreria iostream
using namespace std; ← Spazio di nomi standard
int main()           ← Intestazione di funzione
{
    ↑               ← Nome della funzione
    ...             ← Istruzioni del corpo della funzione
}
```

```
#include  Direttiva al preprocessore
#define   Macro al preprocessore
```

*Definizioni globali*

- Funzioni o Prototipi di Funzioni
- Variabili

*Funzione principale **main***

```
main ()
{
    Dichiarazioni locali
    Istruzioni
}
```

*Definizioni di altre funzioni*

```
funz1 (...)
{
    ...
}
funz2 (...)
{
    ...
}
...
```


# struttura generale di un programma in C++

# Primo programma in C++

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World";

    return 0;
}
```



Tutte le istruzioni che iniziano con # vengono chiamate direttive e vengono processate da un preprocessore.


La direttiva #include indica al compilatore di includere un file, in questo caso lo standard iostream che è la libreria per le operazioni di I/O più comuni.

# Primo programma in C++

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World";

    return 0;
}
```



Import dei nomi definiti nel namespace 'std' (standard). In questo modo importeremo nel nostro codice tutte le definizioni di questo namespace (es. cout).

Senza questa istruzione avrei dovuto scrivere std::cout per richiamare l'istanza cout definita dentro il namespace std.


# Primo programma in C++

```
#include <iostream>
using namespace std;

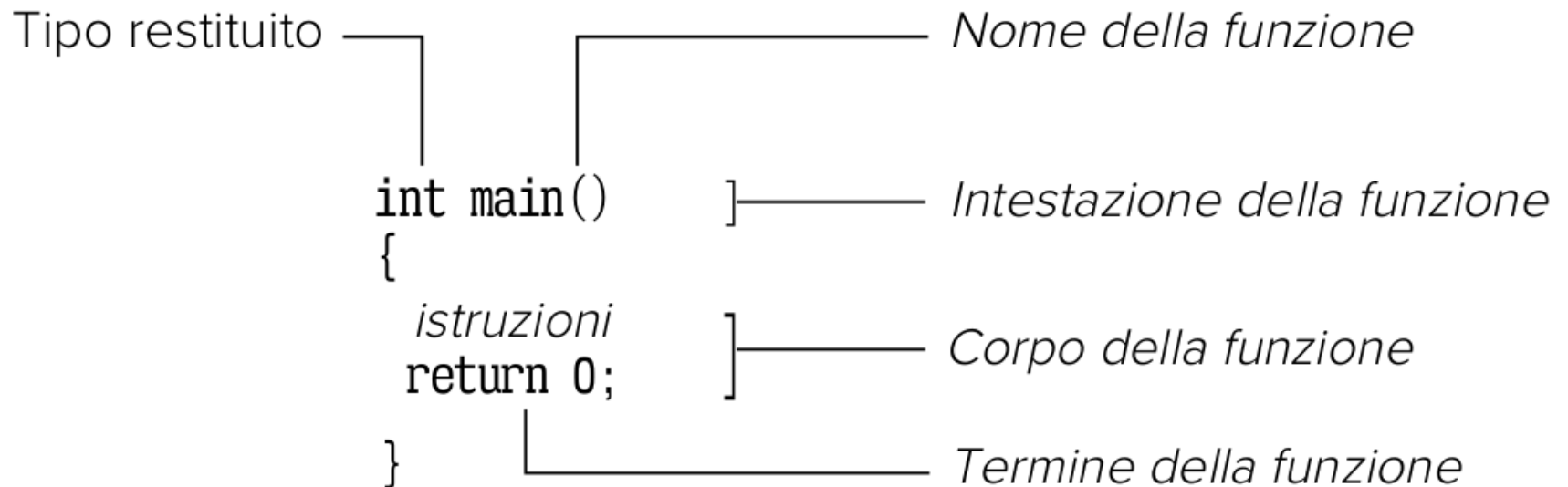
int main()
{
    cout << "Hello World";

    return 0;
}
```

Stampa il messaggio sullo schermo (standard output).



# La funzione `int main()`



*Nota:* le istruzioni terminano con il punto e virgola  
`return 0;` termina la funzione `main()`



# Debugging di un programma in C++

Un programma non gira quasi mai la prima volta che si tenta di compilarlo.

In generale gli errori possono essere di tre tipi:

1. Errori *sintattici* nell'uso delle regole grammaticali del linguaggio sorgente.
2. Errori *logici* nel progetto dell'algoritmo
3. Errori *logici* nella traduzione dell'algoritmo (corretto) in un programma sorgente (scorretto)

# Elementi di un programma in C++

identificatore: sequenza di caratteri che serve ad indicare un dato in memoria o una funzione.

Il primo carattere di un identificatore non può essere una cifra.

Il C++ è distingue fra lettere maiuscole e minuscole.

parole riservate

asm	double	mutable	struct
auto	else	namespace	switch
bool	enum	new	template
break	explicit	operator	this
case	extern	private	throw
catch	float	protected	try
char	for	public	typedef
class	friend	register	union
const	goto	return	unsigned
continue	if	short	virtual
default	inline	signed	void
delete	int	sizeof	volatile
do	long	static	wchar_t
			while

# Tipi di dato predefiniti in C++

Tipo	Dimensione in bytes	Minimo ... Massimo
char	1	-127 ÷ 127 oppure 0 ÷ 255
unsigned char	1	0 ÷ 255
signed char	1	-127 ÷ 127
int	4	-2147483648 ÷ 2147483647
unsigned int	4	0 ÷ 4294967295
signed int	4	-2147483648 ÷ 2147483647
short int	2	-32768 ÷ 32767
unsigned short int	2	0 ÷ 65,535
signed short int	2	-32768 ÷ 32767
long int	8	-2,147,483,648 ÷ 2,147,483,647
signed long int	8	-2,147,483,648 ÷ 2,147,483,647
unsigned long int	8	0 ÷ 4,294,967,295
long long int	8	-(2 <sup>63</sup> ) ÷ (2 <sup>63</sup> )-1
unsigned long long int	8	0 ÷ 18,446,744,073,709,551,615
float	4	
double	8	
long double	12	
wchar_t	2 o 4	1 carattere UNICODE

# Sequenze di “escape” in C++

Sequenza di Escape	Significato	Codici ASCII			
		Dec		Hex	
\n	Nuova riga	13	10	0D	0A
\r	Ritorno di carrello	13		0D	
\t	Tabulazione orizzontale	9		09	
\v	Tabulazione verticale	11		0B	
\a	Bip sonoro	7		07	
\b	Retrocessione di uno spazio	8		08	
\f	Avanzamento di una pagina	12		0C	
\0	Zero, nullo	0		0	
\\	Sbarra inclinata inversa	92		5C	
\'	Apice	39		27	
\"	Virgolette	34		22	
\?	Punto interrogativo	34		22	
\000	Numero ottale	<i>Tutti</i>		<i>Tutti</i>	
\xhh	Numero esadecimale	<i>Tutti</i>		<i>Tutti</i>	

# Variabili e costanti in C++

Una variabile è una sequenza di una o più celle di memoria caratterizzata da un **indirizzo** (quello della prima cella) e da un **tipo**. L'indirizzo è associato dal compilatore al nome della variabile (un **identificatore**), mentre la variabile stessa viene utilizzata per ospitare **valori** di quel tipo che possono poi essere modificati. Le variabili possono ospitare ogni tipo di dato: stringhe, numeri e strutture.

Una costante, invece, è una variabile il cui valore non può essere modificato.

# Inizializzazione di variabili e costanti in C++

## inizializzazione di variabile

`<tipo di dato> <nome variabile> = <espressione>`

**dove** `<espressione>` è qualunque espressione valida il cui valore sia dello stesso tipo che `<tipo di dato>`

## inizializzazione di costante

**const** `<tipo di dato> <nome costante> = <espressione>`

# Visibilità di variabili in C++

La zona di un programma in cui una variabile è attiva si dice visibilità (**scope**) della variabile.

Le variabili in C++ possono essere:

- **locali**: sono quelle definite all'interno di un blocco di istruzioni, tipicamente quello di una funzione, e sono visibili dal punto in cui sono definite fino alla fine di quel blocco
- **globali**: sono quelle che si dichiarano al di fuori di qualunque blocco, quindi anche al di fuori dalla `main()`, e sono visibili dappertutto (quindi da qualunque funzione), meno che nei blocchi (ovvero nelle funzioni) in cui esistono variabili locali con lo stesso nome

# Istruzione di assegnamento

Alle variabili definite e visibili si da un valore tramite l'*istruzione di assegnamento*, ovvero l'operatore =

*variabile = espressione;*



# Costanti

- *costanti letterali*

- *costanti intere* 23 45 ...
- *costanti carattere* 'a' ... 'z'
- *costanti in virgola mobile* 23.14 45.34 ...
- *costanti stringa* "Ancona"

- *costanti enumerative*

```
enum Colori {Rosso,Arancione,Giallo,Verde,Blu,Viola};
```

- *costanti definite*

```
#define NUOVARIGA '\n'  
#define PIGRECO 3.141592  
#define VALORE 54
```

- *costanti dichiarate*

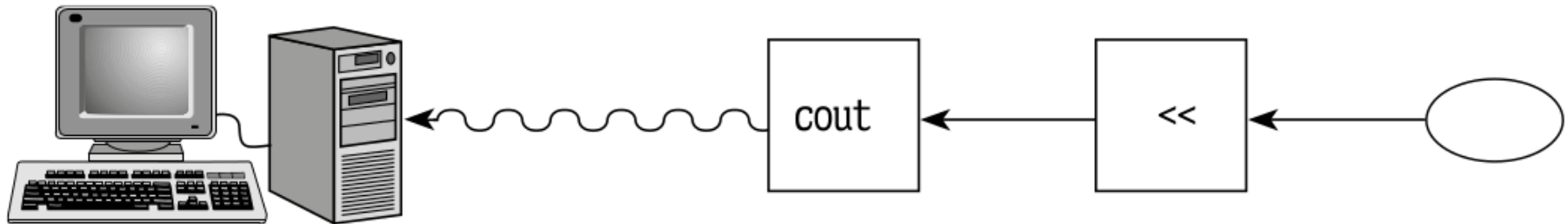
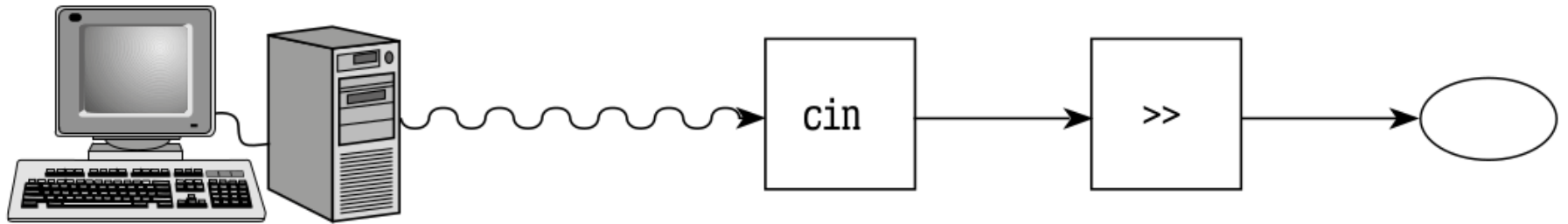
```
const int mesi=12;
```

# Quiz OOP #1



<https://kahoot.com/>

# Input/Output da console



# Input/Output da console

C++ gestisce le operazioni di IO mediante il concetto di flusso (**stream**). Un flusso è una astrazione utile a modellare le comunicazioni tra le applicazioni (i processi) e l'ambiente (il SO e l'hardware).

In C++ le librerie per gli stream sono implementate mediante gerarchie di classi che permettono l'indipendenza del codice per la gestione delle operazioni di IO dallo specifico dispositivo (es. stampante, video, rete, tastiera, file, ecc.).

Gli stream si possono collegare di volta in volta ad entità specifiche (es. tastiera, schermo, file, ..).

# Input/Output da console

Standard output predefinito (oggetto `std::cout`) generalmente associato al video.

```
#include <iostream>
using namespace std;
cout << "Inserire un numero:" << endl;
```

`<<` è l'operatore di inserimento che "spinge" i dati alla sua destra nel canale di output ***cout***, che è di tipo `ostream`.

***endl*** inserisce un ritorno a capo

È necessario includere lo header `iostream`.

# Input/Output da console

Standard input predefinito (oggetto `std::cin`) generalmente associato alla tastiera.

```
#include <iostream>
using namespace std;
int x;
cin >> x;
```

l'operatore di estrazione `>>` viene usato per leggere i dati dallo standard input e direzionarli verso la variabile `x`.

# Input/Output da console

`cout`, `cin`, `cerr` sono ***oggetti*** ed hanno scope (visibilità) globale: il programmatore può usarli all'interno del suo programma senza dover operare inizializzazioni di sorta.

La clausola `using namespace std` permette di risolvere i nomi (classi, funzioni, costanti) definiti nel namespace `std` senza dover usare l'operatore risolutore di scope (ovvero scrivere `std::cout`).

# Input/Output da console

Cosa succede se l'utente inserisce una sequenza di caratteri che non rappresenta un numero e proviamo a convertirlo in un intero o un float? → nella variabile non sarà copiato alcun valore (Errore di IO).

```
float x;  
cout << "Inserisci un numero: ";  
cin >> x;
```

In questo caso, il metodo **fail()** di **cin** restituirà true e l'input buffer verrà mantenuto in uno stato di errore. Per questo occorre chiamare **cin.clear()** per svuotare il buffer per permettere input successivi. **cin.ignore()** serve ad indicare che vogliamo ignorare l'input errato acquisito fino alla generazione dell'errore.



# Input/Output da console

```
float x ;
cout << "Inserisci x: ";
cin >> x;
if(cin.fail()){ // IO error !
    cerr << "IO e r r o r" << endl;
    cin.clear() ; //RESET IO flags
}
else
    cout << "Input: " << x << endl;
cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

**cin.clear()** pone a zero tutti i flag di errore dello stream. Non si potrà procedere con altre operazioni di IO senza aver prima posto a zero i flags di errore.

**cin.ignore()** permette di scartare caratteri rimasti nello stream.

# Input/Output da console

```
string input;  
getline(cin, input);  
while(!cin.fail() && !cin.eof()) {  
    cout << "Frase inserita: " << input << endl;  
    getline(cin, input);  
}
```

`eof()` indica la lettura del carattere EOF (CTRL+d da tastiera (linux) oppure CTRL+z (Win)).

# Le stringhe

Una stringa è una sequenza di caratteri che termina con un carattere speciale `'\0'` che rappresenta la sua fine.

In C++ si ha a disposizione la **classe string**, che permette di manipolare le stringhe agevolmente, come ad esempio l'inserimento automatico del carattere di fine stringa.

```
#include <string> // Header necessario!
string name="Pippo";
string your_name;

cout << "My name is " << name << endl;
cout << "Please write your name: " << endl;
cin >> your_name;
cout << "Your name is" << your_name << endl;
```

# Le stringhe

```
#include <string> // Header necessario!  
string s="abcdef";  
  
cout << "lunghezza di s: " << s.length() << endl;  
cout << s.substr(1, s.length()-1) << endl;
```

`length()` restituisce il numero di caratteri di una stringa.

`substr(x,y)` estrae una sottostringa restituendo un oggetto `string` indicando l'indice del primo carattere (`x`) e la lunghezza della sottostringa (`y`). Se il secondo parametro non viene specificato saranno considerati tutti i rimanenti caratteri fino alla fine della stringa originale.

# Le stringhe

```
#include <sstream>
using namespace std;
stringstream ss;

ss << "Hello world" << endl;
// ... Altre operazioni
cout << ss.str();
```

La classe `stringstream` si può usare per immagazzinare dati (stringhe e numeri) in un buffer da usare successivamente.

`stringstream` si usa in entrambi i versi: estrazione (`>>`) e inserimento (`<<`);

# IO di stringhe

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    string nome, nomeEcognome;
    cout << "Inserisci il tuo nome: ";
    cin >> nome;
    cout << "Ciao " << nome << "!" << endl;
    cin.ignore(); // svuota il buffer di input
    cout << "Inserisci nome e cognome separati da spazio: ";
    getline(cin, nomeEcognome);
    cout << "Ciao " << nomeEcognome << "!" << endl;}
```

l'operatore >> termina la lettura quando trova uno spazio; per leggere una sequenza di caratteri che comprende spazi bianchi si deve utilizzare la funzione di libreria `getline()` invece che l'operatore >>.

# La libreria <cstring>

• <b>strcpy()</b>	• <b>char* strcpy(char* destinazione, const char* sorgente)</b> Copia la stringa <i>sorgente</i> nella stringa <i>destinazione</i> . Restituisce <i>destinazione</i> .
• <b>strncpy()</b>	• <b>char* strncpy(char* destinazione, const char* sorgente, size_t num)</b> Copia <i>num</i> caratteri da <i>sorgente</i> a <i>destinazione</i> . Restituisce <i>destinazione</i>
• <b>strlen()</b>	• <b>size_t strlen (const char* s)</b> Restituisce la lunghezza della stringa <i>s</i> .
• <b>strcat()</b>	• <b>char* strcat(char* destinazione, const char* sorgente)</b> Aggiunge una copia della stringa <i>sorgente</i> alla fine di <i>destinazione</i> . Restituisce <i>destinazione</i> .
• <b>strncat()</b>	• <b>char *strncat(char* s1, const char* s2, size_t n)</b> Aggiunge i primi <i>n</i> caratteri di <i>s2</i> a <i>s1</i> . Restituisce <i>s1</i> . Se <i>n</i> >= <i>strlen(s2)</i> , allora <i>strncat(s1, s2, n)</i> ha lo stesso effetto che <i>strcat(s1, s2)</i> .
• <b>strcmp()</b>	• <b>int strcmp(const char* s1, const char* s2)</b> Confronta le stringa <i>s1</i> ed <i>s2</i> e restituisce: 0 se <i>s1</i> = <i>s2</i> <0 se <i>s1</i> < <i>s2</i> >0 se <i>s1</i> > <i>s2</i>
• <b>stricmp()</b>	• <b>int stricmp(const char* s1, const char* s2)</b> come <i>strcmp()</i> ma <i>case insensitive</i>
• <b>strncmp()</b>	• <b>int strncmp(const char* s1, const char* s2, size_t n)</b> come <i>strcmp()</i> ma solo sui primi <i>n</i> caratteri.
• <b>strnicmp()</b>	• <b>int strncmp(const char* s1, const char* s2, size_t n)</b> come <i>strncmp()</i> ma <i>case insensitive</i> .
• <b>strrev()</b>	• <b>char* strrev(char* s)</b> inverte la stringa passata come argomento.
• <b>strupr()</b>	• <b>char* upr(char* s)</b> mette in maiuscolo la stringa passata come argomento.
• <b>strlwr()</b>	• <b>char* strlwr(char* s)</b> mette in minuscolo la stringa passata come argomento.
• <b>strchr()</b>	• <b>const char* strchr(const char* str, int c)</b> Restituisce un puntatore alla prima occorrenza del carattere <i>c</i> in <i>s</i> . Restituisce NULL se <i>c</i> non è in <i>s</i> .

# La libreria <cstring>

<ul style="list-style-type: none"><li>• <b>strspn()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>size_t strspn(const char* s1, const char* s2)</b> Restituisce la lunghezza della sottostringa più lunga di s1 che comincia con s1[0] e contiene unicamente caratteri presenti in s2</li></ul>
<ul style="list-style-type: none"><li>• <b>strstr()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>const char* strstr(const char* s1, const char* s2)</b> Cerca la stringa s2 in s1 e restituisce un puntatore al carattere dove comincia s2</li></ul>
<ul style="list-style-type: none"><li>• <b>strcspn()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>size_t strcspn(const char* s1, const char* s2)</b> Restituisce la lunghezza della sottostringa più lunga di s1 che comincia con s1[0] e non contiene alcuno dei caratteri presenti in s2</li></ul>
<ul style="list-style-type: none"><li>• <b>strpbrk()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>const char *strpbrk(const char* s1, const char* s2)</b> Restituisce l'indirizzo della prima occorrenza in s1 di qualunque dei caratteri di s2. Restituisce NULL se nessuno dei caratteri di s2 appare in s1</li></ul>
<ul style="list-style-type: none"><li>• <b>memcpy()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>void* memcpy(void* s1, const void* s2, size_t n)</b> Rimpiazza i primi n bytes di *s1 con i primi n bytes di *s2. Restituisce s1</li></ul>
<ul style="list-style-type: none"><li>• <b>strnset()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>char* strnset(char* s, int ch, size_t n)</b> Utilizza strcmp() su una stringa esistente per fissare n bytes della stringa al carattere ch</li></ul>
<ul style="list-style-type: none"><li>• <b>strtok()</b></li></ul>	<ul style="list-style-type: none"><li>• <b>char* strtok(char* s1, const char* s2)</b> Divide la stringa s1 in sottostringhe delimitate dai caratteri presenti nella stringa s2. Dopo la chiamata iniziale strtok(s1, s2), ogni chiamata successiva a strtok(NULL, s2) restituisce un puntatore alla successiva sottostringa in s1. Queste chiamate cambiano la stringa s1, rimpiazzando ogni separatore con il carattere NULL ('\0')</li></ul>



# IO di stringhe

Molto spesso bisogna convertire stringhe di cifre nei corrispettivi formati numerici. C++ fornisce per questi scopi le funzioni `atoi`, `atof` e `atol`.

```
char s1[] = "314";      char s2[] = "3.14";      char s3[] = "-3.14";

cout << "s\t" << "atoi(s)\t" << "atof(s)\t" << "atol(s)\n";
cout << "s1\t" << atoi(s1)
      << "\t" << atof(s1) << "\t" << atol(s1) << endl;
cout << "s2\t" << atoi(s2) << "\t"
      << atof(s2) << "\t" << atol(s2) << endl;
cout << "s3\t" << atoi(s3) << "\t"
      << atof(s3) << "\t" << atol(s3) << endl;
```

s	atoi(s)	atof(s)	atol(s)
s1	314	314	314
s2	3	3.14	3
s3	-3	-3.14	-3

# Generare numeri casuali

```
#include <cstdlib> // per rand() e srand()
#include <ctime> // per la funzione time()

srand(123);
// oppure
srand(time(NULL));

for(int i=0;i<1000;i++)
    cout << rand() << endl;
```

Questo codice genera 1000 numeri casuali tra 0 e RAND\_MAX.

La funzione rand() estrae il prossimo numero dalla sequenza casuale.

srand() fissa il seme (seed). Questo causa la generazione della stessa sequenza di numeri pseudo-casuali.

time() restituisce i secondi trascorsi dal 1/1/1970.

# Generare numeri casuali

```
#include <cstdlib> // per rand() e srand()
#include <ctime> // per la funzione time()

srand(123);
unsigned int r;

for(int i=0;i<1000;i++){
    r = rand()%(b-a+1) + a;
    cout << r << endl;}
```

Questo codice genera 1000 numeri casuali nell'intervallo  $[a,b]$ . Quello seguente invece genera numeri casuali in virgola mobile in  $[0,1]$ .

```
#include <cstdlib> // per rand() e srand()
#include <ctime> // per la funzione time()

srand(123);
double r;

for(int i=0;i<1000;i++){
    r = ((double)rand())/RAND_MAX;
    cout << r << endl;}
```

# Esercizio

Scrivere un programma che richiede all'utente quale operazione svolgere tra quelle elencate (specificando il suo numero) e successivamente la esegue:

1. Simula l'andamento di una partita a carta forbice sasso, determinando le scelte dei due contendenti (giocatore A e giocatore B) in modo casuale. Vince chi fa meglio in N tentativi, con N dato dall'utente. Il valore di N deve essere dispari.
2. Scrivere un programma che simula l'esito della somma del lancio di due dadi, visualizzare il risultato di N lanci dopo aver letto N dalla tastiera.
3. Uscire dal programma.