Capitolo 2

pag.33-64

Il linguaggio C++. Elementi base

Presenta: Prof. Misael Mongiovì

Installazioni per C++

1) Linux

Build essential (normalmente già installati)

```
sudo apt-get install build-essential
```

2) Windows

Windows Subsystem for Linux https://learn.microsoft.com/en-us/windows/wsl/install

Oppure MinGW per operare direttamente dal Prompt dei comandi (cmd)

3) Mac

Xcode da istallare per avere i comandi del compilatore sul terminale

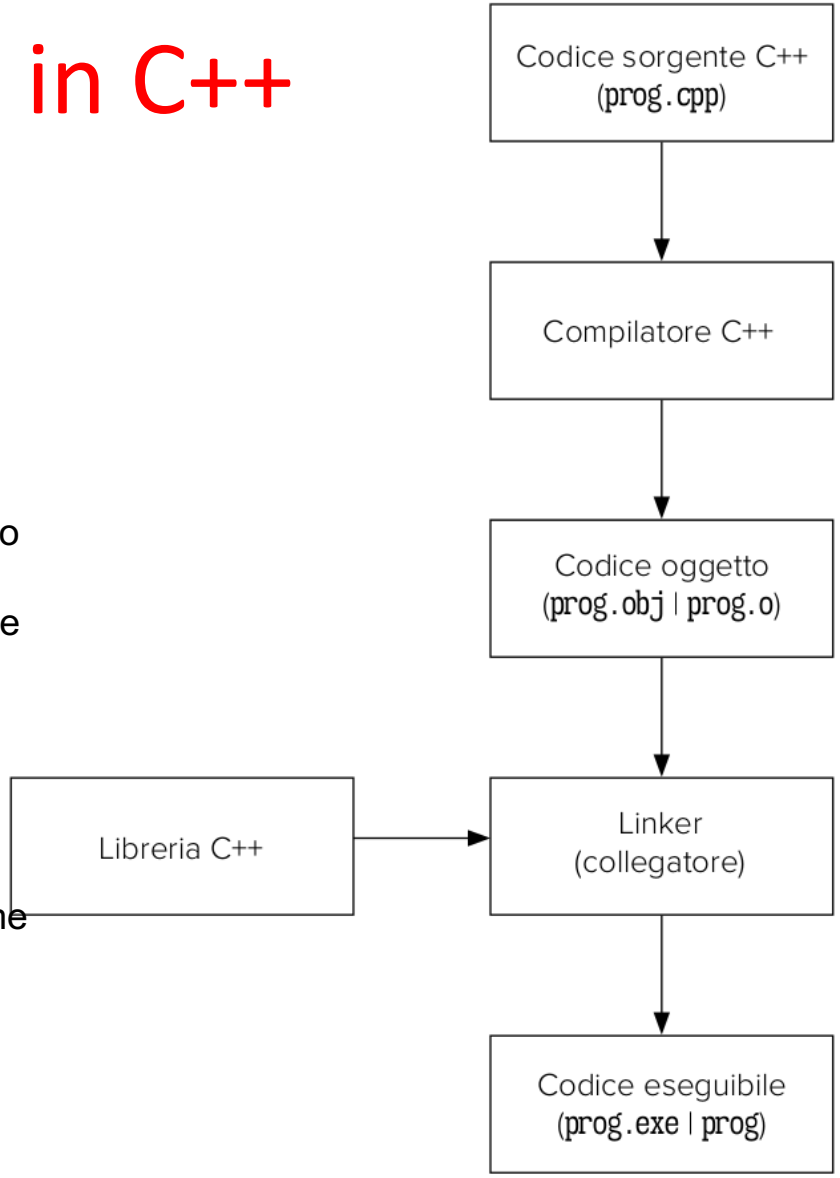
Opereremo da linea di comando. Possiamo scrivere il codice con un editor di testo qualsiasi, ad es. `vi` su terminale linux.

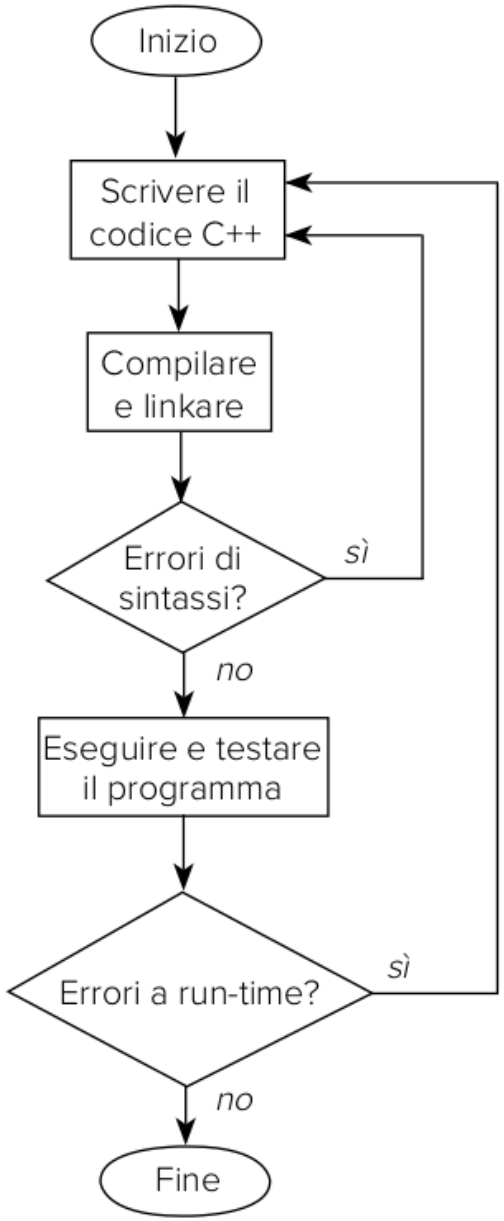
Potete usare IDE (ad es. NetBeans, Eclipse, Xcode su Mac, DevC++ su Windows, Kdevelop su Linux) ma per superare l'esame sarà necessario saper compilare ed eseguire su linea di comando.

costruzione di un programma in C++

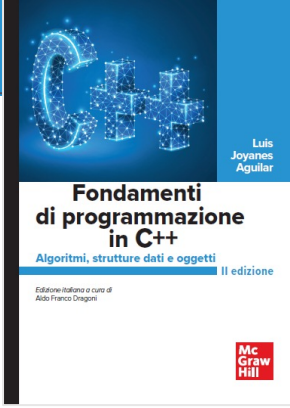
1) editore di testo per scrivere il programma sorgente e salvarlo in un file chiamato file sorgente o codice sorgente. Per convenzione, i programmi scritti in C++ hanno l'estensione .cpp, mentre quelli scritti in C hanno l'estensione .c.
2) Compilare il codice sorgente. Il compilatore traduce il codice 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 una svariate funzioni che realizzano compiti comuni, come la gestione dell'Input/Ouput 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

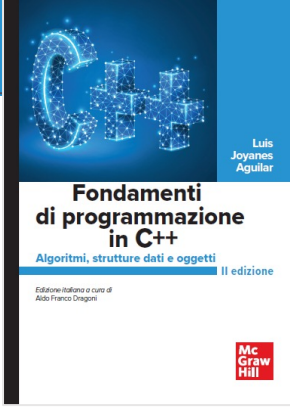
Funzione principale **main**

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

Definizioni di altre funzioni

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

struttura generale di un programma in C++



la funzione `int main()`

Tipo restituito — *Nome della funzione*

```
int main()
{
  istruzioni
  return 0;
}
```

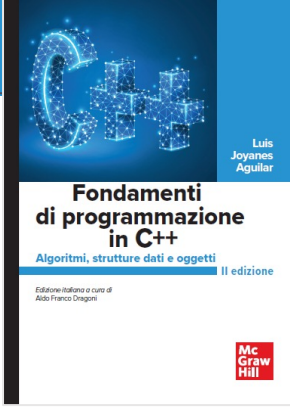
Intestazione della funzione

Corpo della funzione

Termine della funzione

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

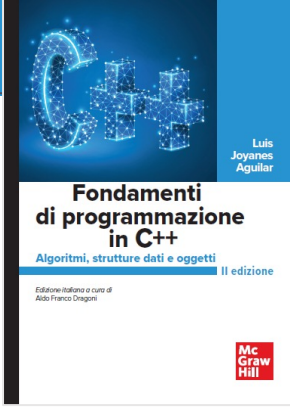
debugging di un programma in C++



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

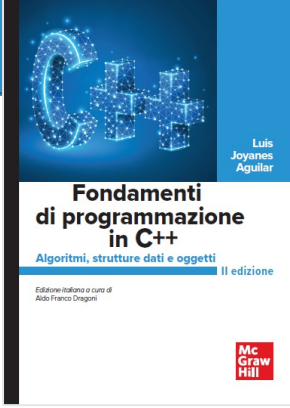
Il primo carattere di un identificatore non può essere una cifra

Il C++ è distingue fra lettere maiuscole e minuscole

parole riservate

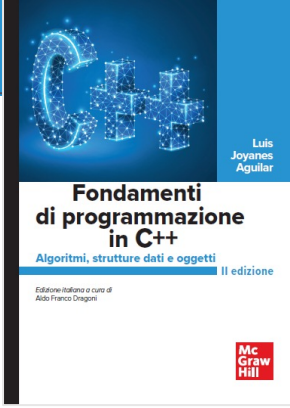
asm	double	mutable	struct
auto	else	namespace	switch
bool	enum	new	template
break	explicit	operator	this
case	extern	private	throw
catch	float	protected	try
char	for	public	typedef
class	friend	register	union
const	goto	return	unsigned
continue	if	short	virtual
default	inline	signed	void
delete	int	sizeof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

Capitolo 2. Il linguaggio C++. Elementi base



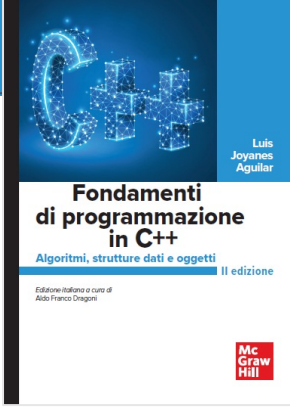
Tipo	Dimensione in bytes	Minimo ... Massimo
char	1	-127 ÷ 127 oppure 0 ÷ 255
unsigned char	1	0 ÷ 255
signed char	1	-127 ÷ 127
int	4	-2147483648 ÷ 2147483647
unsigned int	4	0 ÷ 4294967295
signed int	4	-2147483648 ÷ 2147483647
short int	2	-32768 ÷ 32767
unsigned short int	2	0 ÷ 65,535
signed short int	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	-(2^63) ÷ (2^63)-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

tipi di dato pre-definiti in C++



sequenze di “escape” in C++

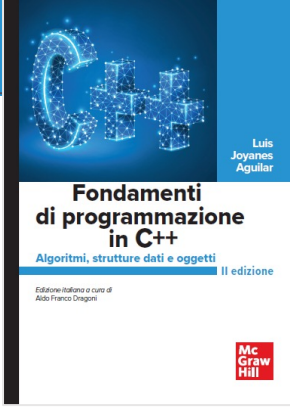
Sequenza di Escape	Significato	Codici ASCII Dec	Codici ASCII Hex
\n	Nuova riga	13 10	0D 0A
\r	Ritorno di carrello	13	0D
\t	Tabulazione orizzontale	9	09
\v	Tabulazione verticale	11	0B
\a	Bip sonoro	7	07
\b	Retrocessione di uno spazio	8	08
\f	Avanzamento di una pagina	12	0C
\0	Zero, nullo	0	0
\\	Sbarra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



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++

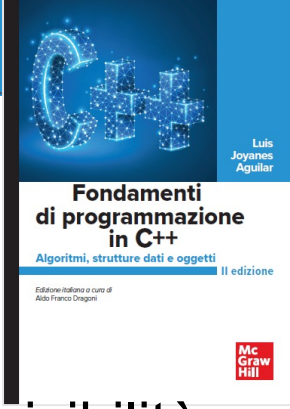
inizializzazione di variabile

```
<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>
```



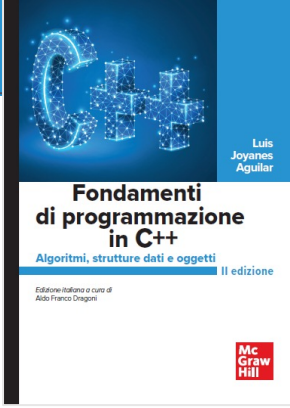
visibilità di variabili in C++

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

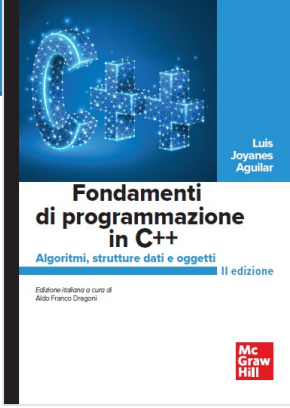
- globali: sono quelle che si dichiarano al di fuori di qualunque blocco, quindi anche al di fuori dalla main(), e sono visibili dappertutto (quindi da qualunque funzione), meno che nei blocchi (ovvero nelle funzioni) in cui esistono variabili locali con lo stesso nome



istruzione di assegnamento

Alle variabili definite e visibili si dà 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
const char * str = "Ancona";
```

Input/Output da console