

# Programmazione a oggetti con C++

```
// Enciphers the message using the  
void encipher(int e, int n, std::string message)
```

1. Le basi del C++
2. Gli array e le stringhe
3. Funzioni, puntatori e strutture di dati
4. La programmazione orientata agli oggetti
5. L'ereditarietà e il polimorfismo
6. La libreria iostream e il lavoro con i file

```
    ciphertext[i] = c;
```

```
}
```

# Le basi del C++

Questo capitolo è un'introduzione all'uso del **linguaggio di programmazione C++** in modalità procedurale:

- ▶ passeremo in rassegna gli **elementi di base del linguaggio**: la struttura del codice, l'uso delle variabili e degli operatori, la gestione dell'input e dell'output;
- ▶ poi vedremo come applicare i principi della **programmazione strutturata** attraverso l'uso delle **strutture di selezione** e i **cicli iterativi**.

## 1.1 La sintassi, le variabili e gli operatori

Per illustrare le regole fondamentali della **sintassi del C++** useremo come esempio il semplice programma della **figura 1.1**, che puoi scaricare dal sito web del libro:

- ▶ salva per prima cosa il file **Terra\_Luna.cpp** in una cartella **corso\_C++** sul tuo desktop; useremo questa cartella per tutti gli esempi del corso;
- ▶ apri poi il file con l'editor di **Dev-C++**, un **ambiente di sviluppo** o **IDE** (*Integrated Development Environment*) open source disponibile per ogni sistema operativo; puoi scaricare Dev-C++ liberamente da Internet.

Le figure di questo libro si riferiscono alla versione 5.11 dell'IDE Dev-C++ usata con il sistema operativo Windows 11.

The screenshot shows the Dev-C++ IDE with the following code in `Terra_Luna.cpp`:

```

1  /* intestazione (header) del codice */
2  #include <iostream>
3  using namespace std;
4  const float c = 299792.458; // velocità della luce nel vuoto, in km/s
5
6  /* funzione principale main(): calcola il tempo necessario
7   alla luce per percorrere la distanza inserita come input */
8  int main(void)
9  {
10     float distanza, tempo;
11
12     cout<<"\n\t Inserisci la distanza media Terra-Luna: ";
13     cin>>distanza;
14
15     tempo = distanza/c;
16
17     cout<<"\n La luce riflessa dalla Luna ci raggiunge in "<<tempo<<" s\n";
18 }
19

```

Annotations in the image:

- A bracket on the right side of lines 1-5 is labeled "intestazione del programma".
- A bracket on the right side of lines 8-18 is labeled "funzione principale main()".

The execution output window shows:

```

Inserisci la distanza media Terra-Luna: 380000
La luce riflessa dalla Luna ci raggiunge in 1.26754 s
-----
Process exited after 15.13 seconds with return value 0
Premere un tasto per continuare . . .

```


**Figura 1.1** Il risultato dell'esecuzione di un semplice programma in C++.

Il **codice sorgente** dei programmi in C++ è un normale file di testo non formattato, che va salvato con l'estensione **.cpp**.


I colori e le formattazioni che vedi nella **figura 1.1** sono stati aggiunti automaticamente dall'editor di Dev-C++ per mettere in evidenza i vari elementi del codice.


## Compilare ed eseguire i programmi

Il C++ è un **linguaggio compilato**: per poter eseguire un programma bisogna convertire il codice sorgente in linguaggio macchina tramite la **compilazione**.

La compilazione di un codice aperto in Dev-C++ si avvia facendo clic sull'icona  oppure premendo il tasto **F9** della tastiera.

Compila ora il codice **Terra\_Luna.cpp** della **figura 1.1**. Vedrai che nella tua cartella **corso\_C++** appare un nuovo file **Terra\_Luna.exe**: questo è il **file eseguibile** in codice binario prodotto dal compilatore.

A questo punto in Dev-C++ puoi eseguire il programma facendo clic sull'icona  oppure premendo il tasto **F10**: si aprirà allora il terminale a linea di comando visibile al piede della **figura 1.1**, dove apparirà l'output del programma.

In Dev-C++ puoi anche compilare ed eseguire in un unico passaggio il codice in lavorazione: basta fare clic sull'icona  o premere il tasto **F11**.

### UN ESERCIZIO AL VOLO

Che cosa viene prodotto dal compilatore del C++?

.....

## La struttura del codice sorgente

Ogni linguaggio di programmazione ha le proprie regole per la stesura del codice.

In C++ il **codice sorgente** inizia con un' **intestazione**, o **header**, che è seguita dalla parte principale del programma sotto forma di **funzione main()**.

Come puoi vedere alle righe 2 e 3 della **figura 1.1**, l'**header** o **intestazione** del codice contiene in particolare:

- ▶ l'indicazione delle **librerie** da includere nel programma, con l'istruzione **#include**, detta **direttiva**; in questo caso la direttiva include la libreria **iostream** del C++, che contiene le funzioni necessarie per i processi di input/output;
- ▶ la definizione del **namespace**, che indica al compilatore dove si trovano le funzioni e gli oggetti a cui il codice fa riferimento; in questo caso alla riga 3 il namespace standard **std** indica l'insieme di funzioni e oggetti del C++ definiti dall'ISO e quindi codificati in modo univoco in qualsiasi computer.

Nella **figura 1.1** fa parte dell'intestazione anche la dichiarazione della costante alla riga 4.

La parte principale del codice si trova poi all'interno della **funzione main()** definita alla riga 8: le **istruzioni** sono quelle che vanno dalla riga 10 alla riga 17, ed è importante notare che sono racchiuse tra le **parentesi graffe** delle righe 9 e 18.

In C++ le **istruzioni** di norma **terminano con il punto e virgola ;** e inoltre i **blocchi di istruzioni** vanno **racchiusi tra parentesi graffe {}**.

Nota che le **indentazioni** delle righe non sono obbligatorie in C++, ma è importante e utile inserirle, perché aiutano a riconoscere più facilmente i diversi blocchi di istruzioni.

È sempre bene aggiungere al codice sorgente **commenti** che ne facilitino la comprensione, come alle righe 1, 4 e 6-7 della **figura 1.1**.

I **commenti** in C++ si scrivono tra i simboli **/\* e \*/**, se sono su più righe, oppure antepo-  
nendo al testo un doppio slash **//**, se sono su un'unica riga.

Il compilatore, al momento di creare il file eseguibile del programma, trascurerà tutto ciò che è scritto in forma di commento.

### UN ESERCIZIO AL VOLO

Quale libreria va inclusa nei codici C++ per gestire input e output?

.....

## Le variabili e le costanti

Una **variabile** è uno spazio di memoria RAM a cui è assegnato un **nome**, un **tipo di dato** e un **valore** che può cambiare nel corso dell'elaborazione.

In C++ **ogni variabile va dichiarata** antepo- nendo al suo nome il **tipo di dato** che le si vuole assegnare. I principali tipi di dato del linguaggio sono elencati nella **tabella 1.1**.

**Tabella 1.1** I principali tipi di dato del C++ e la loro occupazione di memoria.

tipo di dato	byte	valori rappresentabili
int	4	numeri interi positivi e negativi
char	1	i 256 caratteri ASCII
bool	1	i valori logici 0 (falso) oppure 1 (vero)
float	4	numeri decimali (cioè a virgola mobile) positivi e negativi

In una stessa istruzione si possono dichiarare più variabili dello stesso tipo, separando i loro nomi con virgole. Così, per esempio, la riga 10 del codice **Terra\_Luna.cpp** dichiara le due variabili **distanza** e **tempo**, entrambe di tipo **float**, cioè numero decimale.

Anche una **costante** è uno spazio di memoria a cui è assegnato un nome e un tipo di dato, ma **ha un valore fissato**, che non può cambiare nel corso dell'elaborazione.

Per **dichiarare una costante** si deve anteporre al tipo di dato la parola-chiave **const**, come alla riga 4 del codice della **figura 1.1**: in questo caso **c** rappresenta la velocità della luce nel vuoto, il cui valore (qui in km/s) non cambia mai.

Nota che le costanti vanno dichiarate nell'intestazione del codice, mentre le variabili di norma si dichiarano all'interno della **funzione main()**.

**Il linguaggio C++ è case-sensitive**: distingue tra lettere minuscole e maiuscole.

Dunque in C++ una variabile chiamata **A** è *diversa* da una variabile chiamata **a**.

Per i **nomi delle variabili** valgono le seguenti regole:

- ▶ possono contenere cifre, ma non devono iniziare con una cifra né contenere spazi bianchi, o caratteri speciali eccetto l'*underscore* **\_**; così, per esempio, **num1** e **num\_1** sono nomi leciti, mentre **1num** e **num 1** non sono permessi;
- ▶ non devono coincidere con parole-chiave del linguaggio; una variabile non può chiamarsi, per esempio, **const** o **int**.

Le **parole-chiave** o **keyword** elencate nella **tabella 1.2** sono i termini che costituiscono il nucleo fondante del linguaggio. Il loro significato è stabilito in modo univoco dalle convenzioni degli standard ISO e si tratta perciò di termini «riservati».

**Tabella 1.2** Le parole-chiave del linguaggio C++.

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

### UN ESERCIZIO AL VOLO

Perché in C++ non si può dare a una variabile il nome **auto**?

.....

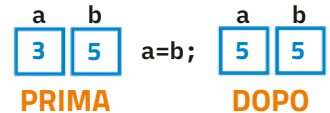
## Inizializzare le variabili: l'operatore di assegnazione =

Per poter usare una variabile (o una costante) in un programma bisogna **inizializzarla**, cioè assegnarle un valore.

Le variabili si possono inizializzare al momento della dichiarazione oppure in seguito. Le costanti invece vanno inizializzate nell'intestazione del codice, quando le si dichiara.

L'inizializzazione più comune usa l'**operatore di assegnazione =** (segno «uguale»), come alle righe 4 e 15 del codice della **figura 1.1**: a destra dell'operatore = si scrive un valore esplicito (riga 4) oppure un'espressione in termini di altre variabili (riga 15).

È importante tenere presente che l'istruzione `a=b;` *non significa* «il valore della variabile a è uguale a quello della variabile b»; significa invece: «cambiamo l'attuale valore della variabile a e le assegniamo il valore che in questo momento ha la variabile b».



## I flussi di input e output

La riga 13 del codice della **figura 1.1** mostra l'altro modo in cui si può inizializzare una variabile: qui infatti `distanza` è inizializzata assegnandole come valore un dato di input inserito dall'utente (nota che `distanza` è dichiarata come `float`, perciò sarà trattata come numero decimale, anche se il dato di input è intero).

In C++ i flussi di dati o **stream di input/output** si gestiscono con le istruzioni `cin` (*character input*) e `cout` (*character output*) con i rispettivi operatori `>>` e `<<`.

Nel caso del codice della **figura 1.1**:

- ▶ alla riga 12 `<<` è l'**operatore di inserimento**: `cout<<` inserisce i dati che seguono nel flusso di output che viene indirizzato allo schermo;
- ▶ alla riga 13 `>>` è l'**operatore di estrazione**: `cin>>distanza;` infatti «estrae» il valore di input digitato sulla tastiera e lo usa per inizializzare la variabile `distanza`.

Una stessa istruzione di output può contenere più flussi distinti, ognuno preceduto dall'operatore di inserimento `<<`. Per esempio nella riga 17 del codice, che scrive a schermo il risultato finale del programma, ci sono tre flussi di output:

```
cout<<"\n La luce riflessa dalla Luna ci raggiunge in "<<tempo<<" s\n";
```

- ▶ il primo e l'ultimo flusso sono racchiusi tra doppi apici, così il compilatore li tratterà come testi;
- ▶ il secondo flusso invece è semplicemente il nome della variabile `tempo`, perciò a schermo verrà mostrato il suo valore (lo stesso varrebbe per un'espressione basata su variabili).

Nelle istruzioni di output della **figura 1.1** merita notare l'uso delle **sequenze di escape** `\n`, per mandare a capo il testo, e `\t`, per inserire tabulazioni.

Come già accennato, il codice può usare `cin` e `cout` grazie alla direttiva `#include <iostream>` dell'intestazione, che include la libreria di input/output del C++ (nota che questa istruzione non richiede il punto e virgola alla fine).

## Gli operatori aritmetici

Gli **operatori aritmetici**, posti tra due variabili, si usano nei calcoli matematici:

+	<i>somma</i>
-	<i>sottrazione</i>
*	<i>moltiplicazione</i>
/	<i>divisione</i>
%	<i>modulo</i>

L'operatore *modulo* restituisce il resto di una divisione tra interi: per esempio, `11%4` darà come risultato 3.

### UN ESERCIZIO AL VOLO

Quale istruzione in C++ dimezza il valore di una variabile `var`?

.....



È importante tenere presente che il risultato delle operazioni può dipendere dal tipo di dato assegnato alle variabili.

La divisione tra interi, per esempio, non restituisce decimali. Così, se *a* e *b* valgono 17 e 4, il rapporto *a/b* varrà 4.25 se le due variabili sono state definite come `float`, mentre varrà 4 se sono state definite come `int`.

L'operatore `%` restituisce il resto della divisione tra interi, perciò si può calcolare *a%b* soltanto quando entrambe le variabili sono state definite come `int` (se così non è, si avrà un errore in fase di compilazione).

Si usano infine le **parentesi tonde** per stabilire la priorità delle operazioni.

Per esempio, per trovare il valore medio tra quelli delle variabili *a* e *b* si dovrà scrivere  $(a+b)/2$ , che ha un valore diverso da  $a+b/2$ .

Quando occorre, si usano parentesi nidificate; per esempio, l'inverso del valore medio appena citato si può scrivere nella forma  $1 / ((a+b) / 2)$ .

## Gli operatori di assegnazione composti

L'operatore di assegnazione si può combinare con quelli aritmetici, producendo «scorciatoie» che permettono di rendere più conciso il codice.

Per esempio, `+=` assegna alla variabile a sinistra dell'operatore la somma del proprio valore e di quello della variabile a destra: scrivere `var1+=var2`; quindi equivale a scrivere `var1=var1+var2`.

In modo analogo funzionano gli operatori `-=`, `*=`, `/=`, e `%=`.

Così, per esempio, se una variabile intera *a* ha il valore 5, dopo l'istruzione `a *= 2`; il suo valore diventerà 10. Dopo un'ulteriore istruzione `a %= 3`; il valore di *a* diventerà 1 (perché il resto della divisione di 10 per 3 è 1).

## Gli operatori relazionali e logici

Gli **operatori relazionali** si usano per confrontare tra loro i valori di due variabili, ottenendo come risultato il valore booleano **1 (vero)** oppure **0 (falso)**:

<code>==</code>	<i>uguale a</i>
<code>!=</code>	<i>diverso da</i>
<code>&gt;</code>	<i>maggiore di</i>
<code>&gt;=</code>	<i>maggiore o uguale a</i>
<code>&lt;</code>	<i>minore di</i>
<code>&lt;=</code>	<i>minore o uguale a</i>

Gli operatori relazionali si usano per formulare le **condizioni** su cui si basano le *strutture di selezione* dei programmi.

La condizione `a==b` per esempio significa: «confronta il valore di *a* con quello di *b*, restituendo **1 (vero)** se sono uguali, altrimenti **0 (falso)**».

Si possono poi combinare tra loro più condizioni usando gli **operatori logici**:

<code>&amp;&amp;</code>	<i>equivale a AND</i>
<code>  </code>	<i>equivale a OR</i>
<code>!</code>	<i>equivale a NOT</i>

Ciò significa che:

- ▶ `condizione1 && condizione2` restituisce **vero** soltanto se entrambe le condizioni sono vere, altrimenti restituisce **falso**;
- ▶ `condizione1 || condizione2` restituisce **vero** se è vera almeno una delle due condizioni, mentre restituisce **falso** soltanto se entrambe le condizioni sono false;
- ▶ `! condizione` restituisce **vero** soltanto se la condizione è falsa, e viceversa.

### UN ESERCIZIO AL VOLO

Se due variabili sono dichiarate come `int a=4, b=3`, quale valore restituirà la condizione `a!=b`?

.....

## Gli operatori di incremento e decremento

Gli **operatori di incremento e decremento** ++ e -- fanno aumentare o diminuire di un'unità il valore di una variabile intera:

- |    |                  |  |
|----|------------------|--|
| ++ | aumenta di 1:    | scrivere <code>i++</code> equivale a scrivere <code>i=i+1</code> |
| -- | diminuisce di 1: | scrivere <code>i--</code> equivale a scrivere <code>i=i-1</code> |

Come vedremo, questi operatori tornano molto utili quando si programmano le strutture ripetitive chiamate *cicli iterativi*.

### RIPASSA I CONCETTI-CHIAVE



- ▶ Ogni programma in C++ ha un'**intestazione** o **header** (con le direttive per l'inclusione delle librerie e l'istruzione che specifica il namespace) seguita da una **funzione** `main()` le cui istruzioni terminano con il **punto e virgola** e vanno racchiuse tra **parentesi graffe**.
- ▶ Il C++ è **case-sensitive**, cioè distingue tra le lettere minuscole e quelle maiuscole.
- ▶ Una **variabile** è un'area di memoria che registra un dato il cui valore può cambiare nel corso dell'esecuzione di un programma. In C++ la **dichiarazione** di una variabile deve specificare il suo **tipo di dato** seguito dal **nome**.
- ▶ L'**inizializzazione** assegna un valore a una variabile, nella dichiarazione o in un'istruzione successiva, tramite l'**operatore di assegnazione** =.
- ▶ Una **costante**, che ha un valore fissato una volta per tutte, va dichiarata e inizializzata nell'intestazione usando la parola-chiave `const`.
- ▶ I principali **tipi di dato** del C++ sono `int` (numero intero), `float` (numero decimale), `char` (carattere) e `bool` (valore booleano).
- ▶ In C++ l'input e l'output si gestiscono mediante gli stream `cin` (con l'**operatore di estrazione** `>>`) e `cout` (con l'**operatore di inserimento** `<<`).
- ▶ L'**operatore di assegnazione** = permette di cambiare il valore registrato negli spazi di memoria corrispondenti alle **variabili** usate nei programmi.
- ▶ Gli altri principali operatori del C++ sono gli **operatori aritmetici** (+ - \* / %), gli **operatori relazionali** (== != > >= < <=), gli **operatori logici** (&& || !) e gli **operatori di incremento/decremento** (++ --).

### ESERCIZI

1. Quale istruzione in C++ dichiara una variabile intera `num` e la inizializza con il valore 7?  
.....
2. La libreria matematica del C++ si chiama `cmath`. Quale direttiva va scritta all'inizio di un codice per poterla utilizzare?  
.....
3. Quale istruzione in C++ acquisisce un valore di input e lo assegna alla variabile `var`?  
.....
4. Quale istruzione in C++ scrive a schermo «Ora `var` vale X», dove X è l'attuale valore della variabile `var`?  
.....
5. Scrivi nel modo più conciso possibile l'istruzione di C++ che cambia il valore di una variabile chiamata `nome_lungo`, triplicandolo.  
.....
6. Scrivi l'espressione che in C++ fa diminuire di un'unità il valore di una variabile intera `num`.  
.....

## 1.2 Le strutture condizionali, o selezioni

Qualsiasi **algoritmo** si può codificare usando tre sole strutture fondamentali: la **sequenza**, la **selezione** o **struttura condizionale** e il **ciclo** o **struttura iterativa**.

Questa proprietà, dimostrata dal celebre *teorema di Böhm-Jacopini* dell'informatica teorica, è alla base del paradigma chiamato **programmazione strutturata**.

Nel precedente esempio della **figura 1.1** il programma conteneva semplicemente una **sequenza** di istruzioni, che venivano eseguite dalla prima all'ultima.

Ora vedremo invece come le **selezioni** e i **cicli iterativi** permettano di far eseguire ai programmi certi blocchi di istruzioni, oppure altri, al verificarsi di opportune **condizioni**.

### Le selezioni a due vie

La struttura condizionale più comune è la **selezione a due vie**, dove il programma esegue istruzioni diverse a seconda del fatto che una condizione sia vera o falsa. A questo scopo si usa la struttura **if...else** (cioè «se... altrimenti...»).



#### LA SINTASSI DELLA STRUTTURA if...else

```
if (condizione)
    {istruzioni da eseguire se la condizione è vera;}
else
    {istruzioni da eseguire se la condizione è falsa;}
```

queste istruzioni (una come minimo) sono sempre presenti

queste istruzioni sono presenti nelle selezioni a due vie

L'istruzione **else**, così come l'istruzione **if**, non deve terminare con il punto e virgola, che va invece messo come sempre alla fine delle altre istruzioni.

Per riassumere, nella struttura **if...else**:

- ▶ se la condizione è vera, il programma eseguirà le istruzioni che vengono dopo **if**;
- ▶ se la condizione è falsa, invece, eseguirà le istruzioni che vengono dopo **else**.

I blocchi delle istruzioni da eseguire vanno racchiusi tra parentesi graffe.

Scrivi e salva nella tua cartella **corso\_C++**, per esempio, il codice **valore\_ass.cpp** della **figura 1.2**. Questo programma restituisce in output il valore assoluto del numero inserito dall'utente come input, usando la struttura condizionale delle righe 10-13:

- ▶ se il dato di input non è negativo, la condizione **numero>=0** è vera e viene eseguita l'istruzione di output della riga 11;
- ▶ altrimenti, se il dato di input è minore di zero, la condizione è falsa e viene eseguita l'istruzione di output della riga 13.

```
valore_ass.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main(void)
5  {
6      float numero;
7      cout<<"\n\t Inserisci un numero qualsiasi: ";
8      cin>>numero;
9      // stampa il valore assoluto del numero inserito come input
10     if (numero>=0)
11         {cout<<"\n il valore assoluto del numero e' "<<numero;}
12     else
13         {cout<<"\n il valore assoluto del numero e' "<<-numero;}
14 }
15
```

condizione sul dato di input

**Figura 1.2** Un programma che stampa il valore assoluto del numero di input.

### UN ESERCIZIO AL VOLO

È possibile che vengano eseguite sia le istruzioni associate a **if**, sia quelle associate a **else**? Perché?



Nota che il programma potrebbe anche essere ridotto a una selezione a una sola via, cioè senza la clausola `else`, scrivendo dopo la riga 8 le due istruzioni:

```
if (numero<0) { numero = -numero; }
cout<<"\n il valore assoluto del numero e' "<<numero;
```

Questo codice funzionerebbe altrettanto bene, però modificherebbe il valore della variabile `numero` se l'utente ha inserito un dato negativo; perciò il dato di input originario non sarebbe più disponibile, nel caso si volessero aggiungere ulteriori elaborazioni.

Molti algoritmi richiedono l'uso di più **selezioni annidate** una nell'altra.

Scarica per esempio dal sito web del libro il codice `multiplo.cpp` della **figura 1.3** e testane il funzionamento.

Questo programma chiede di inserire due numeri interi positivi e poi controlla se il maggiore tra i due numeri sia o meno multiplo del minore.

Per il controllo è ideale usare l'operatore aritmetico modulo `%`, che restituisce 0 quando la divisione di un intero per un altro non dà resto.

Bisogna tuttavia prestare attenzione al fatto che l'operatore funziona correttamente soltanto quando i due numeri interi sono entrambi positivi, e inoltre non sappiamo quale sarà il maggiore tra i due dati di input inseriti dall'utente.

```

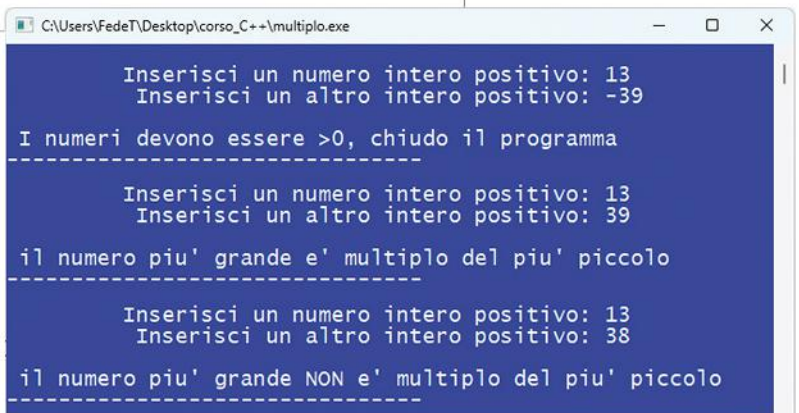
multiplo.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main(void)
5  {
6      int numA, numB;
7      cout<<"\n\t Inserisci un numero intero positivo: ";
8      cin>>numA;
9      cout<<"\t Inserisci un altro intero positivo: ";
10     cin>>numB;
11     // stabiliamo anzitutto che nessuno dei due numeri sia zero o negativo
12     if (numA <= 0 || numB <= 0) {
13         cout<<"\n I numeri devono essere >0, chiudo il programma "; }
14     else { // scopriamo se il numero più grande è multiplo di quello più piccolo
15         if (numA >= numB) {
16             if (numA%numB==0)
17                 {cout<<"\n il numero piu' grande e' multiplo del piu' piccolo";}
18             else
19                 {cout<<"\n il numero piu' grande NON e' multiplo del piu' piccolo";} }
20         else {
21             if (numB%numA==0)
22                 {cout<<"\n il numero piu' grande e' multiplo del piu' piccolo";}
23             else
24                 {cout<<"\n il numero piu' grande NON e' multiplo del piu' piccolo";} } }
25     }
26

```

**Figura 1.3**  
Un programma con strutture di selezione annidate.

condizione composta sui due dati di input

nota come l'**indentazione** dei blocchi di istruzioni aiuti a identificare visivamente il progressivo annidamento delle strutture condizionali



Il codice della **figura 1.3** contiene selezioni annidate una nell'altra su tre livelli:

- ▶ **riga 12**: il primo `if` controlla, con una condizione composta che usa l'operatore logico OR, se uno o l'altro dei numeri di input sia negativo: se è così, il programma invia il messaggio della riga 13 e poi termina;
- ▶ **riga 15**: se entrambi i numeri sono positivi, questo `if` annidato controlla se il primo numero sia maggiore del secondo: se è così, l'ulteriore selezione annidata delle **righe 16-19** determina se il primo numero è multiplo del secondo e stampa a schermo il responso;
- ▶ **riga 20**: il percorso `else` dell'`if` annidato viene seguito se il primo numero è *minore* del secondo: in tal caso l'ulteriore selezione annidata delle **righe 21-24** determina se il secondo numero è multiplo del primo e stampa a schermo il responso.

## L'istruzione `else if`

Quando ci sono più di due eventualità di cui tenere conto, si può usare `else if`.

Ogni istruzione `else if` permette infatti di specificare una nuova condizione da controllare.

Scarica per esempio dal sito web del libro il codice `semaforo.cpp` della [figura 1.4A](#) e testane il funzionamento inserendo diversi caratteri come input.

Il programma indica che cosa fare quando è accesa la luce del semaforo specificata come input (trascuriamo per semplicità il caso di più luci accese simultaneamente).

Le luci possibili sono tre, ma si deve tenere conto anche dell'eventualità che l'utente inserisca un carattere diverso da quelli consentiti, nel qual caso non si può rispondere.

**Figura 1.4A** Un programma che usa l'istruzione condizionale `else if`.

```
semaforo.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main(void)
5  {
6      char luce;
7      cout<<"\n\t Di che colore e' la luce del semaforo?";
8      cout<<"\n\t scrivi V per verde, G per giallo o R per rosso: ";
9      cin>>luce;
10
11     if (luce == 'V' || luce == 'v')
12     { cout<<"\n\t\t Puoi passare "; }
13     else if (luce == 'G' || luce == 'g')
14     { cout<<"\n\t\t FERMATI se puoi farlo in sicurezza "; }
15     else if (luce == 'R' || luce == 'r')
16     { cout<<"\n\t\t DEVI FERMARTI! "; }
17     else
18     { cout<<"\n\t ATTENZIONE: dato di input non valido "; }
19 }
20
```

nota che le condizioni imposte nelle istruzioni `if` ed `else if` devono escludersi a vicenda



```
C:\Users\FedeT\Desktop\corso_C++\semaforo.exe
Di che colore e' la luce del semaforo?
scrivi V per verde, G per giallo o R per rosso: r
DEVI FERMARTI!
-----
Di che colore e' la luce del semaforo?
scrivi V per verde, G per giallo o R per rosso: V
Puoi passare
-----
Di che colore e' la luce del semaforo?
scrivi V per verde, G per giallo o R per rosso: F
ATTENZIONE: dato di input non valido
-----
```

Le quattro eventualità possibili sono controllate nel codice aggiungendo, oltre ai percorsi `if` ed `else`, anche due `else if` con le relative istruzioni da eseguire.

Nota che nelle condizioni alle righe 11, 13 e 15:

- ▶ poiché `luce` è una variabile di tipo `char`, i valori usati per i confronti devono essere scritti tra apici;
- ▶ le condizioni composte consentono all'utente di inserire l'input indifferentemente come lettera maiuscola oppure minuscola (che sono caratteri diversi per il C++).

## La struttura `switch`

Una possibile alternativa all'uso di `else if` è la struttura condizionale `switch`.

L'istruzione `switch` permette di controllare se una variabile abbia uno tra i valori di un elenco, e in tal caso esegue le istruzioni corrispondenti a quel valore.

**La sintassi della struttura switch**

```
switch (variab)
{
    case valore1_elenco:
        istruzioni da eseguire se variab==valore1_elenco;
        break;
    case valore2_elenco:
        istruzioni da eseguire se variab==valore2_elenco;
        break;
    .....
    default:
        istruzioni da eseguire se il valore di variab è diverso da tutti i valore_elenco;
}
```

La parola-chiave **case** significa «nel caso in cui il valore (della variabile o dell’espressione data come argomento di **switch**) sia uguale a». Ciascun blocco di istruzioni associato a un valore **case** termina con l’istruzione **break**. Le istruzioni **case** devono terminare con i due punti, le istruzioni **break** con il punto e virgola.

Va tenuto presente che nella struttura **switch** si possono associare le stesse istruzioni a più valori dell’elenco di confronto, riunendo i relativi **case**.

Nella **figura 1.4B**, per esempio, puoi vedere una struttura **switch** che equivale alle selezioni del codice precedente. Come utile esercizio, modifica quel codice inserendovi questa struttura, salvalo con il nome **semaforo.cpp** e verifica che funzioni correttamente.

```
11 switch(luce) {
12     case 'V':
13     case 'v':
14         cout<<"\n\t\t Puoi passare ";
15         break;
16     case 'G':
17     case 'g':
18         cout<<"\n\t\t FERMATI se puoi farlo in sicurezza ";
19         break;
20     case 'R':
21     case 'r':
22         cout<<"\n\t\t DEVI FERMARTI! ";
23         break;
24     default:
25         cout<<endl<<"\n\t ATTENZIONE: dato di input non valido"; }
```

**Figura 1.4B** Una struttura **switch** equivalente alle selezioni della **figura 1.4A**.

**RIPASSA I CONCETTI-CHIAVE**



- ▶ La **struttura condizionale** (o **selezione**) **if...else** consente di eseguire istruzioni diverse a seconda che la **condizione** data risulti vera oppure falsa.
- ▶ Le strutture di selezione possono essere **annidate** una dentro l’altra.
- ▶ Quando le eventualità possibili sono molte, si può usare l’istruzione **else if** oppure il costrutto basato sull’istruzione **switch**.

**ESERCIZI**

1. Scrivi una condizione che è vera soltanto quando la variabile **var** ha un valore compreso tra 0 e 1 (estremi inclusi) oppure vale 10.  
.....
2. Scrivi un programma che mostri a schermo «vittoria in casa» se l’utente inserisce come input 1, «pareggio» se inserisce X e «vittoria in trasferta» se inserisce 2. Salva il programma come **esito\_1X2.cpp**.  
.....

## 1.3 I cicli iterativi

Il terzo elemento caratteristico della **programmazione strutturata**, insieme alle sequenze di istruzioni e alle strutture condizionali, sono i cicli iterativi.

Le **strutture iterative**, o **cicli iterativi**, permettono di ripetere un gruppo di istruzioni fintantoché una data **condizione** rimane vera.

Le strutture iterative aiutano a scrivere codici sorgente semplici e chiari e sono un meccanismo fondamentale nella definizione degli algoritmi di tipo matematico.

*Iterare* significa «ripetere» e infatti con una struttura iterativa:

- ▶ si esegue più volte, ciclicamente, un’istruzione o un gruppo di istruzioni, fino a quando la condizione specificata è vera;
- ▶ quando poi la condizione diventa falsa, il ciclo termina e il programma prosegue con l’esecuzione delle istruzioni successive.

In C++ esistono **tre diversi tipi di struttura iterativa**: sono chiamati ciclo di tipo **for**, di tipo **while** e di tipo **do** (detto anche **do-while**).

### Il ciclo di tipo for

Il ciclo di tipo **for** si usa quando il numero delle iterazioni da fare è prefissato, cioè lo si conosce prima dell’inizio del ciclo.

L’istruzione **for** dunque fa ripetere un dato insieme di istruzioni fino a quando rimane vera una condizione che diventa falsa dopo un numero ben preciso di iterazioni.

#### LA SINTASSI DEL CICLO for

```
for (inizializzazione di var; condizione; aggiornamento di var)
{
    istruzioni da eseguire finché la condizione è vera;
}
```

la variabile (qui chiamata **var**) può avere un nome qualsiasi

le istruzioni del ciclo vanno racchiuse tra parentesi graffe

Nella sintassi dell’istruzione **for** appare la variabile-contatore **var**: è quella che viene aggiornata durante il ciclo, per conteggiare le iterazioni già fatte.

Alla fine dell’istruzione **for** non si mette il punto e virgola. Invece bisogna ricordare di separare con punti e virgola i tre argomenti di **for** racchiusi tra le parentesi tonde:

- ▶ il primo argomento assegna il valore iniziale alla variabile-contatore;
- ▶ il secondo argomento contiene la condizione da soddisfare perché il ciclo si ripeta;
- ▶ il terzo argomento aggiorna il valore del contatore **var** alla fine di ogni iterazione.

Ecco che cosa succede quando nel codice è presente un’istruzione **for**:

1. per prima cosa viene assegnato il valore iniziale a **var**, poi inizia il ciclo;
2. viene controllata la verità della condizione;
3. se la condizione è vera, vengono eseguite le istruzioni interne al ciclo, poi viene aggiornato il valore di **var** e si torna al punto 2;
4. altrimenti, cioè se la condizione è falsa, si esce dal ciclo.

Di solito l’aggiornamento della variabile è un incremento, cioè si fa aumentare a ogni iterazione il suo valore di una quantità prestabilita.

Scrivi e testa per esempio il codice **inflazione1.cpp** della **figura 1.5**, che calcola l’aumento del costo della vita per un tasso di inflazione dato come input.

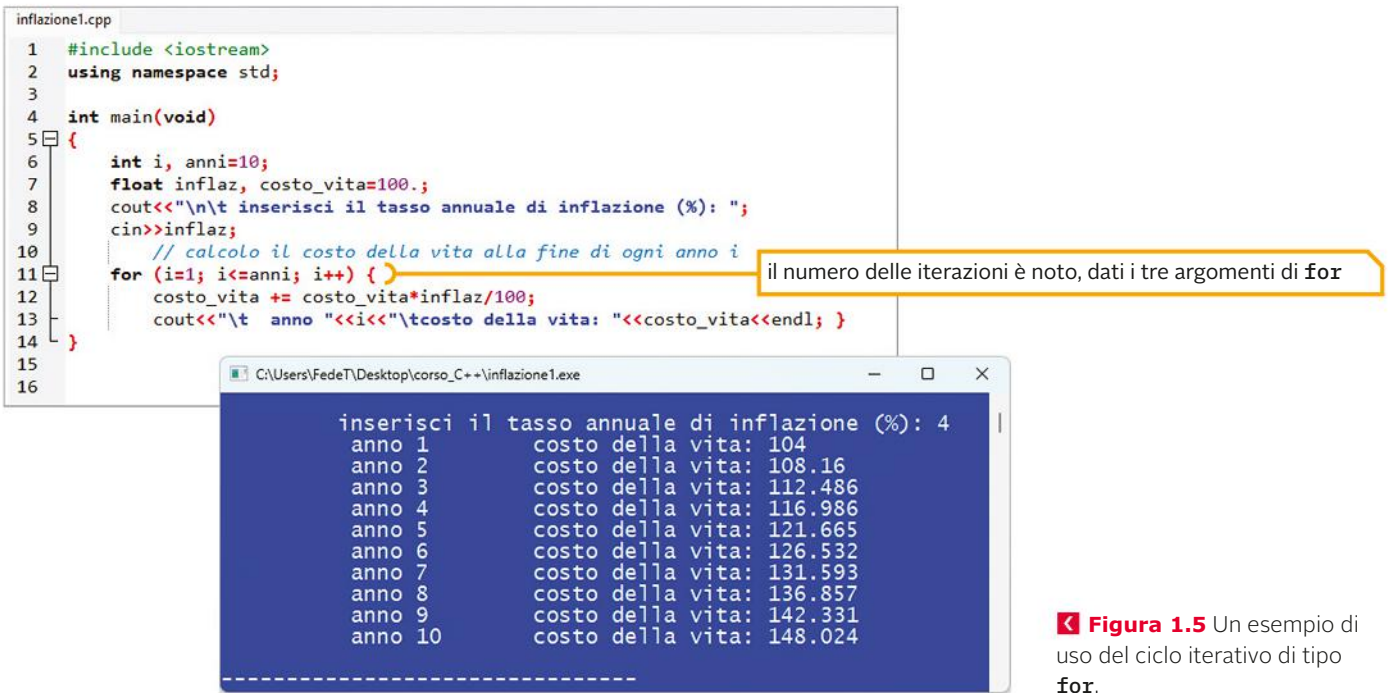
Alla riga 7 la variabile **costo\_vita** è inizializzata con il valore 100. Quindi la riga 8 chiede all’utente di inserire un valore numerico (inteso come percentuale) che la riga 9 assegna alla variabile **inflaz**.

#### UN ESERCIZIO AL VOLO

Quale istruzione **for** fa ripetere un ciclo per i soli valori *dispari* del contatore **i** compresi tra 1 e 15?

.....





◀ **Figura 1.5** Un esempio di uso del ciclo iterativo di tipo for.

Il programma poi usa il ciclo `for` delle righe 11-13 per mostrare a schermo il costo della vita nei dieci anni successivi:

- ▶ nell’istruzione `for` della riga 11 la variabile contatore `i` in questo caso varia da 1 a 10, con incremento di un’unità a ogni iterazione;
- ▶ nota alla riga 12 l’uso dell’operatore composto `+=`: il valore di `costo_vita` viene aumentato a ogni iterazione della percentuale del suo valore precedente dovuta all’inflazione;
- ▶ alla riga 13 la variabile contatore `i` è usata anche per la stampa dei risultati.

Nota che in questo algoritmo il numero delle iterazioni è predefinito, quando ha inizio il ciclo, dal valore che è stato assegnato nella riga 6 alla variabile `anni`.

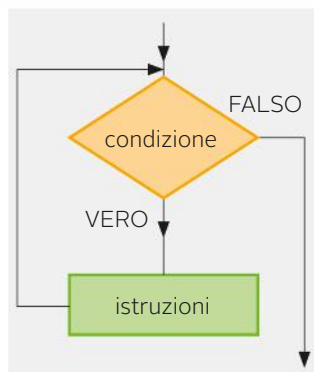
Quando invece il numero delle iterazioni non è noto in anticipo, non si può usare il ciclo `for` ma bisogna ricorrere a un ciclo di tipo `while` oppure di tipo `do-while`.

## Il ciclo di tipo while

Il **ciclo di tipo while** si usa se il numero delle iterazioni da fare non è prestabilito e ha la seguente sintassi:

```

inizializzazione di var
while (condizione su var)
    {istruzioni da eseguire finché la condizione è vera;}
    
```



### UN ESERCIZIO AL VOLO

Quante iterazioni fa di norma un ciclo di tipo `while`?

.....



Nota che l'istruzione `while` non richiede il punto e virgola (il compilatore lo interpreterebbe come fine del ciclo, generando probabilmente un loop infinito).

La variabile `var` che viene usata per esprimere la condizione dev'essere inizializzata esternamente al ciclo; il suo valore viene poi aggiornato a ogni iterazione come parte delle istruzioni del ciclo.

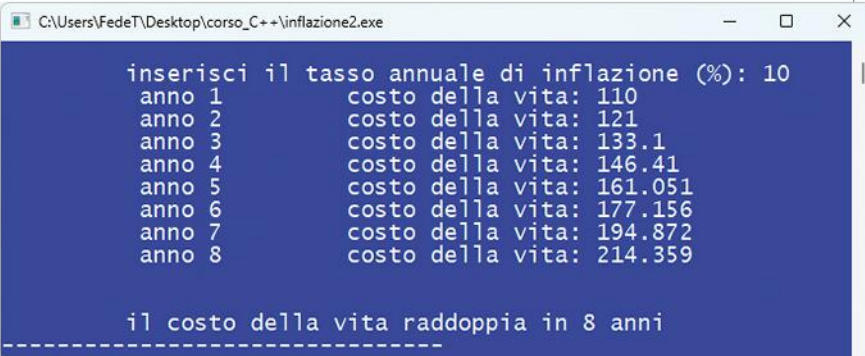
Supponiamo, per esempio, di voler sapere entro quanti anni il costo della vita raddoppierà, per un dato tasso di inflazione.

Il calcolo da fare sarà simile a quello del programma della [figura 1.5](#), ma ora non possiamo sapere quante iterazioni saranno necessarie per ottenere il risultato: il loro numero, infatti, dipenderà dal valore del tasso di inflazione inserito dall'utente.

Si deve usare perciò un ciclo di tipo `while`, come nel codice `inflazione2.cpp` che puoi vedere nella [figura 1.6](#).

Scrivi anche questo programma, modificando il precedente, e testalo provando diversi valori per l'input.

```
inflazione2.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main(void)
5  {
6      float inflaz, costo_vita=100.;
7      cout<<"\n\t inserisci il tasso annuale di inflazione (%): ";
8      cin>>inflaz;
9      // calcolo il costo della vita fino a quando supera il 200%
10     int anno=0;
11     while (costo_vita<200.) {
12         costo_vita += costo_vita*inflaz/100;
13         anno++;
14         cout<<"\t anno "<<anno<<"\t costo della vita: "<<costo_vita<<endl; }
15     cout<<endl<<"\n\t il costo della vita raddoppia in "<<anno<<" anni";
16 }
17
```



**Figura 1.6** Un esempio di uso del ciclo iterativo di tipo `while`.

il numero delle iterazioni non è prevedibile, dipenderà dal valore del dato di input `inflaz`

Anche qui il valore della variabile `costo_vita` è inizialmente posto uguale a 100 ed è poi aggiornato dalla riga 12 a ogni iterazione. La riga 13 incrementa a ogni iterazione il contatore degli anni.

Quando il valore di `costo_vita` supera 200, cioè se nell'ultimo anno considerato è diventato più che doppio rispetto al valore iniziale, all'iterazione successiva la condizione della riga 11 risulta falsa, perciò il ciclo termina.

A questo punto il controllo passa all'istruzione di output della riga 15, che è esterna al ciclo. Questa istruzione mostra a schermo il valore che la variabile `anno` ha assunto durante l'ultima iterazione del ciclo.

Il programma della [figura 1.6](#) funziona bene per dati di input positivi, ma (verificalo!) entrerà in un `loop` infinito se l'utente inserisce come input il valore zero o un valore negativo per il tasso di inflazione: in tal caso infatti il valore di `costo_vita` non aumenterà con le iterazioni del ciclo, quindi non potrà mai raggiungere la soglia 200.

In situazioni come questa è dunque bene impostare un controllo sull'input inserito dall'utente, per assicurarsi che sia compatibile con l'algoritmo. Una buona strategia è inserire l'acquisizione del dato di input all'interno di una struttura iterativa; a questo scopo è particolarmente indicato, come ora vedremo, l'uso del ciclo di tipo `do`.

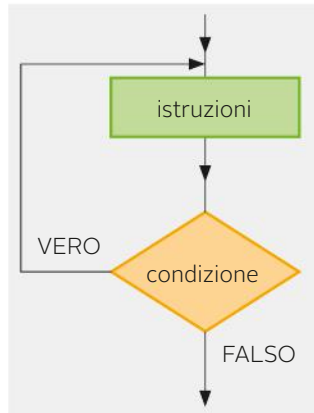
### UN ESERCIZIO AL VOLO

Se il valore del dato di input aumenta, come cambierà il numero delle iterazioni del codice della [figura 1.6](#)?

## Il ciclo di tipo do

Il **ciclo di tipo do** (o **do-while**) è equivalente a quello di tipo **while**, ma ha la condizione posizionata *dopo* le istruzioni, con la seguente sintassi:

```
do
    {istruzioni da eseguire finché la condizione è vera;}
while (condizione su var);
```



Questo ciclo si usa dunque quando si vuole essere sicuri che le sue istruzioni siano eseguite almeno una volta (cioè prima del primo controllo della validità della condizione).

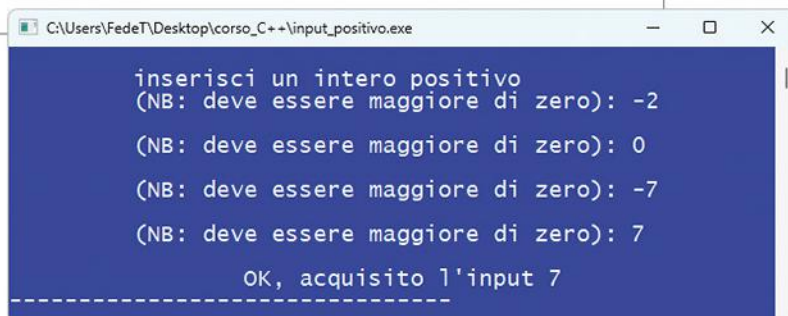
È importante notare che l'istruzione **while**, quando fa parte del ciclo **do**, indica la fine della struttura e quindi deve terminare con il punto e virgola, diversamente da ciò che accade nel ciclo **while**.

Scrivi per esempio il codice **input\_positivo.cpp** della **figura 1.7** e analizzane il funzionamento. Questo semplice programma chiede all'utente di inserire un numero intero positivo, che poi mostra a schermo. Se però l'utente inserisce un valore negativo o nullo, il programma non usa il dato e ripropone invece la richiesta iniziale.

L'istruzione di input **cin** del ciclo certamente verrà eseguita almeno la prima volta. Se l'utente inserisce valori non permessi (zero o numeri negativi), l'istruzione **while** della riga 13 farà ripetere il ciclo fino a quando non viene inserito un dato valido.

```
input_positivo.cpp
1  #include <iostream>
2  using namespace std;
3
4  int main(void)
5  {
6      int dato_input;
7      cout<<"\n\t inserisci un intero positivo ";
8      // ripete la richiesta di inserimento finché l'input è positivo
9
10     do {
11         cout<<"\n\t (NB: deve essere maggiore di zero): ";
12         cin>>dato_input;
13     }
14     while (dato_input<=0);
15
16     cout<<"\n\t\t OK, acquisito l'input "<<dato_input;
17 }
```

queste due istruzioni verranno certamente eseguite almeno una volta



**Figura 1.7**  
Un esempio di uso del ciclo di tipo **do-while**.

### UN ESERCIZIO AL VOLO

Quante iterazioni fa di norma un ciclo **do-while**?

.....

## I cicli annidati

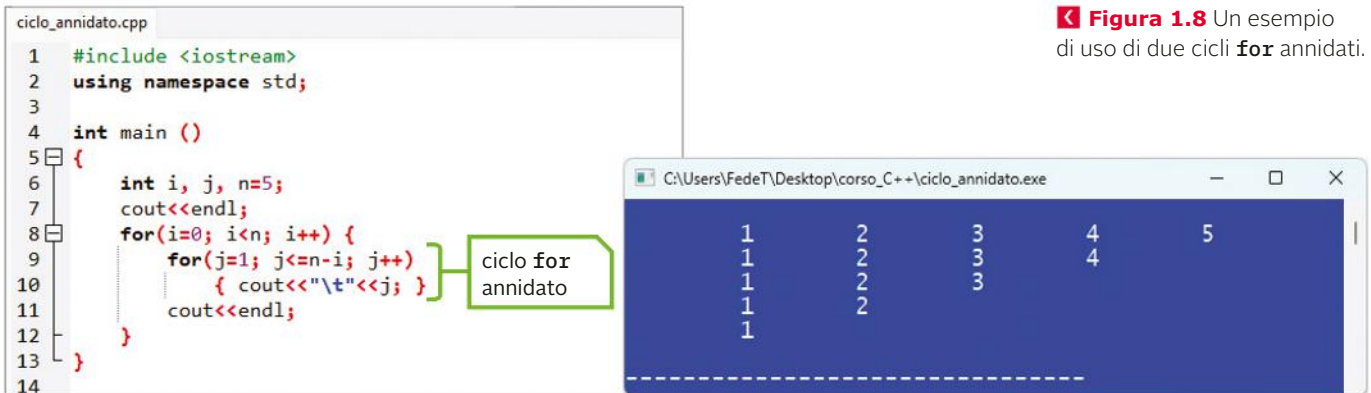
Come le selezioni, anche i cicli iterativi possono essere **annidati** uno nell'altro.

L'output «triangolare» del programma della **figura 1.8**, per esempio, è stato prodotto da due cicli di tipo **for** inseriti uno all'interno dell'altro.

Osserva attentamente il codice per capire bene il suo funzionamento:

- ▶ il **ciclo esterno**, che inizia alla riga 8 e termina alla riga 12, esegue un numero di iterazioni pari al valore della variabile intera  $n$ , incrementando a ogni iterazione il valore del contatore  $i$ ;
- ▶ a ogni iterazione del ciclo esterno viene eseguito l'intero ciclo interno delle righe 9-10, che ha come contatore  $j$  e stampa una riga di numeri separati da tabulazioni;
- ▶ poiché il ciclo interno termina quando il contatore  $j$  raggiunge il valore  $n-i$ , a ogni iterazione del ciclo esterno vengono stampati sulla riga meno numeri;
- ▶ all'ultima iterazione del ciclo esterno  $i$  vale  $n-1$ , perciò nel ciclo interno la condizione di uscita è  $j \leq n-n+1$ , cioè  $j \leq 1$ ; siccome  $j$  inizia dal valore 1, dopo un'unica iterazione il programma termina.

Nota che l'istruzione della riga 11, che manda a capo dopo la stampa di ogni riga di numeri, fa parte del ciclo esterno, non di quello interno.



**Figura 1.8** Un esempio di uso di due cicli **for** annidati.

## RIPASSA I CONCETTI-CHIAVE



- ▶ Le **strutture iterative**, o **cicli iterativi**, permettono di ripetere un gruppo di istruzioni fintantoché una data **condizione** rimane vera.
- ▶ Il **ciclo di tipo for** si usa quando il numero di iterazioni da eseguire è prestabilito.
- ▶ Se il numero delle iterazioni non è noto all'inizio del ciclo, si usa il **ciclo di tipo while** (oppure il **ciclo di tipo do-while**, che esegue sempre almeno una iterazione).

## ESERCIZI

1. Scrivi il ciclo di tipo **for** che stampa a schermo l'elenco dei primi 10 interi positivi in ordine inverso, cioè da 10 a 1.  
.....
2. Scrivi un ciclo di tipo **while** equivalente al ciclo **for** dell'esercizio precedente.  
.....  
.....
3. Modifica il codice della **figura 1.6** in modo tale che non possa entrare in un loop infinito, qualunque sia il dato di input inserito dall'utente. Salva il programma modificato con il nome **inflazione3.cpp**.
4. Modifica il codice della **figura 1.7** in modo tale che funzioni allo stesso modo, ma usando un ciclo di tipo **while** anziché ciclo di tipo **do**. Salva il programma modificato con il nome **input\_positivo\_while.cpp**.

# Informatica e società



## I LINGUAGGI PIÙ USATI DAI PROGRAMMATORI

Il sito web **Stack Overflow** è un forum dove programmatori di tutto il mondo possono porre domande e scrivere risposte riguardo a questioni di ogni genere legate allo sviluppo software.

Con oltre 20 milioni di utenti registrati, Stack Overflow è diventato un importante punto di riferimento per la comunità internazionale degli sviluppatori.

Ogni anno il sito propone ai suoi utenti un dettagliato questionario riguardo alle tecnologie usate, al modo di lavorare e alle esigenze più diffuse.

Decine di migliaia di programmatori rispondono ogni anno al questionario e i risultati dettagliati sono liberamente accessibili nel sito [survey.stackoverflow.co](https://survey.stackoverflow.co).

Qui esaminiamo alcuni dati relativi all'uso dei diversi linguaggi di programmazione nell'anno 2023.

I due grafici in basso mostrano le risposte degli sviluppatori di professione. I dati indicano la percentuale di citazioni per ciascun linguaggio (ogni sviluppatore ne usa più di uno).

I linguaggi risultati più popolari in assoluto (grafico di sinistra) hanno carattere specialistico: i primi quattro della classifica erano nell'ordine **JavaScript** (un linguaggio del tutto distinto da Java, nonostante il nome simile), **HTML/CSS** e **TypeScript**, usati per lo sviluppo di applicazioni web, e **SQL**, usato per la gestione dei database relazionali.

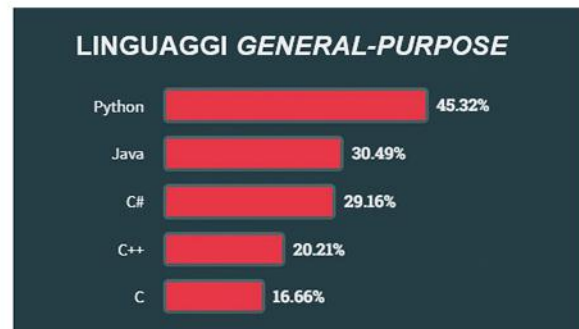
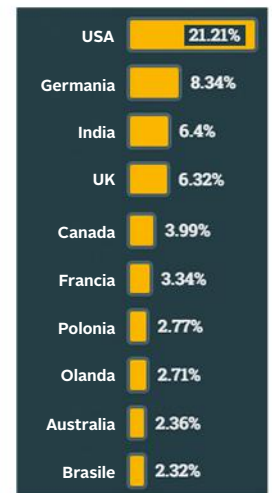
Tra i **linguaggi general-purpose**, cioè utilizzabili per applicazioni di ogni genere, la classifica era quella mostrata dal grafico a barre riprodotto in basso a destra, al piede della pagina.

Da questo grafico si riconosce che:

- il linguaggio di gran lunga più popolare è **Python**, che ha avuto un'impetuosa crescita negli ultimi anni;
- **Java**, che fino a pochi anni fa dominava questa classifica, rimane un linguaggio molto diffuso;
- rimangono importanti anche i linguaggi **C/C++** e **C#** (che si pronuncia *si-sharp*), quest'ultimo soprattutto tra i programmatori professionisti.

Qui a fianco puoi vedere la distribuzione geografica degli sviluppatori che hanno risposto al questionario.

Il predominio statunitense forse non sorprenderà, ma è interessante notare la presenza importante dell'India, dove oggi avviene una parte significativa dello sviluppo software mondiale.



### RIFLETTICI SU

1. Quali Paesi europei stanno assumendo un ruolo sempre più rilevante nella fornitura di servizi software? Suggerimento: puoi dedurlo dalla distribuzione geografica dei programmatori che hanno risposto al questionario di Stack Overflow.

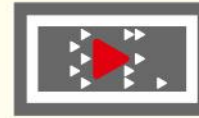
.....  
 .....  
 .....

2. I linguaggi più usati dagli sviluppatori si basano tutti sul paradigma della programmazione a oggetti, ma c'è un'importante eccezione: il linguaggio C, che è procedurale. Sai immaginare perché questo linguaggio è ancora così diffuso?

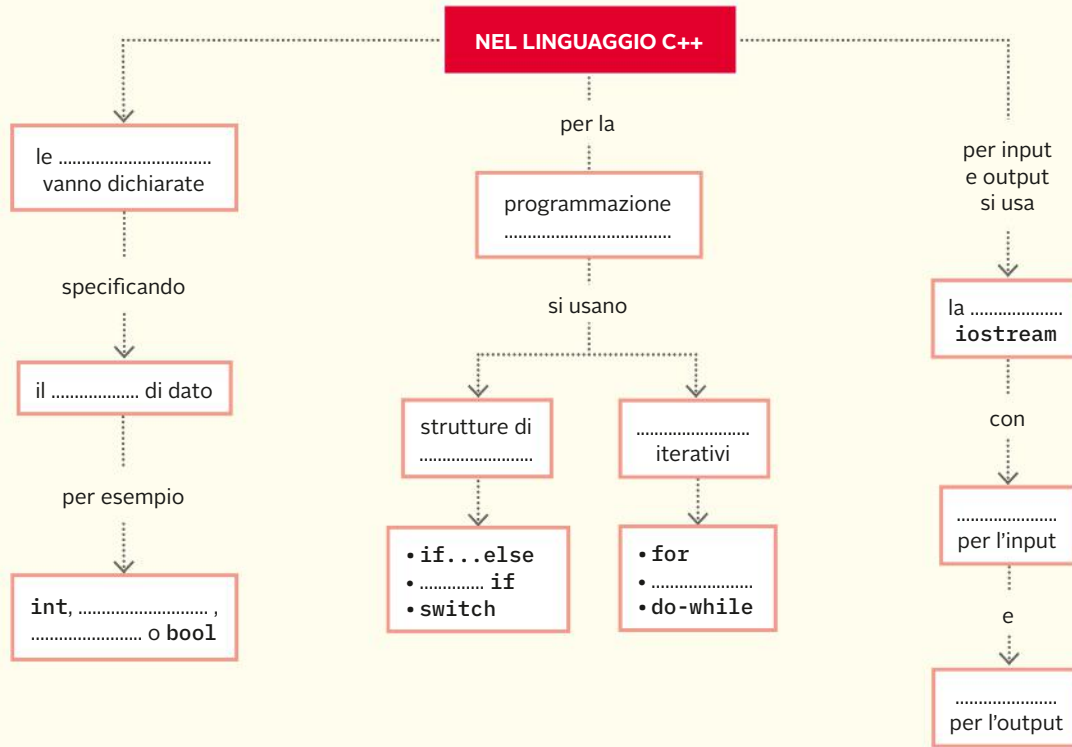
.....  
 .....  
 .....



# ESERCIZI DI RIEPILOGO



**1** Completa la mappa con le parole:  
 cicli • char • cin • cout • else • float • libreria • selezione • strutturata • tipo • variabili • while



**2** Che cosa si intende quando si dice che un linguaggio è *case-sensitive*?

.....  
 .....

**3** Quale tra questi *non* è un tipo di dato in C++?

- A bool                       C float  
 B var                          D int

**4** Come si scrive in C++ la condizione che restituisce il valore booleano 1 (vero) quando il valore della variabile **var1** è uguale a quello della variabile **var2**?

.....

**5** Se la variabile **var1** vale 12 e la variabile **var2** vale 3, qual è il valore restituito dall'operazione **var1%var2**?

.....

**6** Scrivi in C++ una condizione che è vera se la variabile **a** è positiva oppure se è maggiore della variabile **b**.

.....

**7** Che cos'è nel linguaggio C++ **iostream**?

.....

**8** In quale tipo di ciclo iterativo si ha la certezza che le istruzioni verranno eseguite almeno una volta?

.....

**9** In quale caso è più opportuno usare il ciclo di tipo **while** anziché quello di tipo **for**?

- A Quando il numero delle iterazioni è grande.  
 B Quando il numero delle iterazioni è piccolo.  
 C Quando il numero delle iterazioni non è predeterminabile.  
 D I due cicli sono del tutto equivalenti.

**10** Quale output produrrà il codice della figura?

.....

```

1 #include <iostream>
2 using namespace std;
3
4 int main (void)
5 {
6     int x=6;
7     int y=3;
8
9     if (x%3!=0 && x/y==2)
10        {cout<<"prima risposta";}
11     else
12        {cout<<"seconda risposta";}
13 }
    
```



**11** Alle variabili intere  $x$  e  $y$  sono stati assegnati rispettivamente i valori 7 e 3. Associa a ciascuna espressione dell'elenco il risultato corrispondente.

- |                      |         |
|----------------------|---------|
| 1. $x != y$          | A. 0    |
| 2. $x \% y$          | B. 8    |
| 3. $y / x$           | C. 1    |
| 4. $x += y$          | D. vero |
| 5. $x = x + (y / 2)$ | E. 10   |
1. ....    2. ....    3. ....    4. ....    5. ....

**12** Quale output produrrà il seguente ciclo iterativo?

```
int j=1;
for (int i=10; i>1; i-=3)
cout<<" iterazione n. "<<j++<<endl;
```

.....  
 .....  
 .....

**13** Il codice `calcola_area` della figura produce come output l'area di un rettangolo quando l'utente inserisce come input le lunghezze dei due lati adiacenti.

Modifica il codice, che puoi scaricare dal sito web del libro, in modo che l'output specifichi anche se la figura geometrica considerata è un rettangolo oppure un quadrato. Salva il file con lo stesso nome.

```
calcola_area.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main (void)
5 {
6     float lato1, lato2;
7     cout<<"inserisci la misura del primo lato: ";
8     cin>>lato1;
9
10    cout<<"inserisci la misura del secondo lato: ";
11    cin>>lato2;
12
13    cout<<endl<<"l'area misura: "<<lato1*lato2<<endl;
14 }
```

**14** Modifica il codice dell'esercizio precedente introducendo due cicli iterativi tali che, se l'utente inserisce un input negativo o uguale a zero, il programma ripeta la richiesta di inserimento fino a quando il valore della lunghezza del lato è positivo. Salva il file con il nome `calcola_area2.cpp`.

**15** Scrivi un programma in C++ che chieda all'utente di inserire come input due numeri decimali e restituisca in output – come nella figura – il numero maggiore e la differenza tra questo e il numero minore.

Salva il file con il nome `differenza.cpp`.

```
inserisci il primo numero decimale: 12.4
inserisci il secondo numero decimale: 12.7
il numero piu' grande e' 12.7
e supera l'altro numero di 0.3
-----
```

**16** Scrivi un programma in C++ che proponga all'utente il quiz mostrato nella figura, con una risposta corretta e tre risposte errate.

Dopo che l'utente ha risposto il programma deve dare un opportuno feedback, qualunque sia la risposta (quindi anche se *non* è un numero tra 1 e 4).

Usa la struttura condizionale `switch`, scrivendo il minor numero di istruzioni possibile.

Salva il file con il nome `switch_quiz.cpp`.

```
C:\Users\FedeT\Desktop\corso_C++\switch_quiz-SOLUZ.exe
Chi ha creato il sistema operativo Linux?
1. Guglielmo Cancelli
2. Stefano Lavori
3. Linus Torvalds
4. Richard Stallman

la tua risposta: 2
risposta errata
-----
```

**17** Scrivi un programma in C++ che, quando l'utente inserisce come input un valore decimale per la base  $b$  e un valore intero per l'esponente  $E$ , restituisca in output il valore della potenza  $b^E$ .

Il codice deve usare il ciclo iterativo di tipo `for` e deve eseguire correttamente il calcolo per tutti i possibili valori di  $E$  (positivi, zero e negativo). Salva il file con il nome `potenza.cpp`.

**18** Scrivi un programma in C++ che abbia le seguenti caratteristiche:

- ▶ chiede all'utente di inserire come primo input un intero  $x$  maggiore di 50, e ripete la richiesta finché ciò non avviene;
- ▶ chiede all'utente di inserire come secondo input un intero  $y$  compreso tra 1 e 10, e ripete la richiesta finché ciò non avviene;
- ▶ calcola e mostra a schermo il valore di  $x$  modulo  $y$  tramite un ciclo iterativo che usa soltanto l'operazione aritmetica di differenza, cioè il segno «meno»;
- ▶ verifica che il risultato sia lo stesso che si ottiene con l'operatore aritmetico modulo, cioè scrivendo l'espressione  $x \% y$ , producendo un output come quello della figura qui sotto.

Verifica che il programma funzioni correttamente anche con input diversi da quelli dell'esempio in figura.

Salva il file con il nome `modulo.cpp`.

```
C:\Users\FedeT\Desktop\corso_C++\modulo-SOLUZ.exe
inserisci un numero x maggiore di 50: 44
inserisci un numero x maggiore di 50: 57
inserisci un numero y compreso tra 1 e 10: 12
inserisci un numero y compreso tra 1 e 10: -3
inserisci un numero y compreso tra 1 e 10: 6

usando l'operatore aritmetico, 57%6 vale: 3
...e anche il programma trova il risultato: 3
-----
```

# VERIFICA DELLE COMPETENZE

## 1 LAVORA CON I DATI

Scrivi in C++ un programma **inverso.cpp** che abbia le seguenti caratteristiche:



- ▶ la funzione `main()` deve assegnare alla variabile `num` il valore intero inserito dall'utente, controllando che non sia uguale a zero;
- ▶ il codice deve concludersi con un'istruzione che esegue un calcolo e stampa il risultato, così che l'output abbia questa forma:

```

C:\Users\FedeT\Desktop\corso_C++\inverso.exe
inserisci un intero diverso da zero: 3
l'inverso di 3 vale 0.333333
-----
    
```

Quale modifica al programma farà sì che possa produrre l'output mostrato qui sotto?

```

C:\Users\FedeT\Desktop\corso_C++\inverso_MODIF.exe
inserisci un numero diverso da zero: 0.4
l'inverso di 0.4 vale 2.5
-----
    
```

## 2 LAVORA SULL'IMMAGINE

```

1 #include <iostream>
2 using namespace std;
3
4 int main (void)
5 {
6     int x=10;
7     int y=4;
8     .....
9     .....
10    .....
11    .....
12    cout<<"\n adesso x vale "<<x<<" mentre y vale "<<y;
13 }
    
```

Completa il codice della figura qui sopra scrivendo, al posto dei puntini, quattro istruzioni aggiuntive in modo tale che il programma produca questo output:

```

adesso x vale 4 mentre y vale 10
-----
    
```

## 3 SCOPRILO SU INTERNET

Cerca in Rete informazioni sul linguaggio C++ e scopri in particolare:



- ▶ quale libreria consente di usare nei codici in C++ la costante matematica  $\pi$  e quale nome le assegna;
- ▶ quale libreria, e con quali funzioni, consente di visualizzare i risultati tramite lo stream `cout` con un numero prefissato di cifre decimali.

Poi scrivi un programma **pigreco\_cifre.cpp** che visualizzi due volte a schermo la costante  $\pi$  del C++, producendo come output prima il valore **3.14** e poi il valore **3.1416**.

## 4 INFORMATICA E CITTADINANZA

EDUCAZIONE CIVICA

Immagina di dover progettare il software che gestisce il funzionamento di una «sbarra intelligente» che all'uscita di un parcheggio immette su una strada trafficata.



La sbarra è dotata di sensori che riconoscono sia l'arrivo di auto dal parcheggio, sia la presenza di traffico sulla strada.

Normalmente la sbarra sarà abbassata, ma dovrà alzarsi quando un veicolo le si avvicina dall'interno del parcheggio (variabile booleana), ma solo se sulla strada non transitano veicoli (altra variabile booleana).

Quale costrutto della programmazione strutturata useresti nel codice, e in che modo?

.....

.....

.....

## 5 PENSACI TU!

Progetta un programma che riceva come input un voto in pagella e produca come output un giudizio, in base alla tabella qui sotto.

voto	giudizio
10	Ottimo
9	
8	Buono
7	Discreto
6	Sufficiente
5	Insufficiente
4	
3	Gravemente insufficiente
2	
1	
0	

Se input dell'utente non è un intero compreso tra 0 e 10, il programma deve mandare un avviso e richiedere un nuovo input, fino a quando il dato inserito è corretto; al terzo input non valido, il programma deve terminare l'esecuzione, dopo averlo comunicato.

Scrivi il codice usando il minor numero possibile di istruzioni, testalo e salvalo con il nome **Pagella.cpp**.

# CLIL

Learn from the originals

Read it as they wrote it

## Floating Point Arithmetic: Issues and Limitations

from a docs.python.org tutorial, October 30, 2023

Floating-point numbers are represented in computer hardware as base 2 (binary) fractions. For example, the **decimal** fraction **0.625** has value  $6/10 + 2/100 + 5/1000$ , and in the same way the **binary** fraction **0.101** has value  $1/2 + 0/4 + 1/8$ . These two fractions have identical values, the only real difference being that the first is written in base 10 fractional notation, and the second in base 2. Unfortunately, most decimal fractions cannot be represented exactly as binary fractions. A consequence is that, in general, the decimal floating-point numbers you **enter** are only approximated by the binary floating-point numbers **actually** stored in the machine.

The problem is easier to understand at first in base 10. Consider the fraction  $1/3$ . You can approximate that as a base 10 fraction: 0.3 or, better, 0.33 or, better, 0.333 and so on. **No matter how many** digits you're willing to write down, the result will never be exactly  $1/3$ , but will be an increasingly better approximation of  $1/3$ .

In the same way, no matter how many base 2 digits you're willing to use, the decimal value 0.1 cannot be represented exactly as a base 2 fraction. In base 2,  $1/10$  is the infinitely repeating fraction

**0.0001100110011001100110011001100110011001100110011001100110011...**

Stop at any **finite** number of bits, and you get an approximation.

On most machines today, floats are approximated using a binary fraction with the numerator using the first 53 bits starting with the most significant bit and with the denominator as a power of two. In the case of  $1/10$ , the binary fraction is **3602879701896397 /  $2^{55}$**  which is close to but not exactly equal to the true value of  $1/10$ .

On most machines, if Python were to print the true decimal value of the binary approximation stored for 0.1, it would have to display:

```
>>> 0.1
0.1000000000000000055511151231257827021181583404541015625
```

That is more digits than most people find useful, so Python keeps the number of digits manageable by displaying a **rounded** value instead:

```
>>> 1 / 10
0.1
```

Just remember, even though the printed result looks like the exact value of  $1/10$ , the actual stored value is the nearest representable binary fraction.

Note that this is in the very nature of binary floating-point: this is not a bug in Python, and it is not a bug in your code either. You'll see the same kind of thing in all languages that support your hardware's floating-point arithmetic.

### Glossary

<b>to enter</b>	to put information into a computer, book, or document
<b>actually</b>	in fact or really
<b>no matter what, when, why, etc.</b>	idiom that emphasizes that something is always true
<b>finite</b>	having a limit or end
<b>rounded [number]</b>	replaced with an approximate value that has a shorter representation

### QUESTIONS

**1** What is the decimal value of the binary fraction 0.111?

.....

**2** What is used in computers to approximate floating point numbers?

.....

**3** If you execute the following C++ code:

```
float A = 0.1 + 0.1 + 0.1;
cout<<(A == 0.3);
```

what will the result be and why?

.....  
 .....  
 .....