Qualsiasi programma in C++ segue lo schema:

```
#include <iostream>  // libreria che gestisce flusso di input e output

using namespace std; // uso di librerie standard del C++

int main()
{
  // dichiarazioni delle variabili utilizzate
  ...
  // istruzioni del programma
  ...
  return 0;
}
```

Variabili

Le istruzioni racchiuse tra parentesi graffe { } costituiscono un **blocco** di elaborazione. La variabili dichiarate all'interno di un blocco sono **locali**, mentre quelle dichiarate fuori da qualsiasi blocco sono **globali**

Tipi di variabili numeriche predefiniti

int, unsigned int, long, unsigned long, float, double, long double

quanto spazio occupano in memoria:

tipo	**Bytes**	**Range**	**Precisione (decimale)**
int	2	–32768 a 32767	..
long	4	$-2x10^9$ a $2x10^9$	..
unsigned int	2	0 a 65535	..
unsigned long	4	0 a $4x\ 10^9$	..
float	4	10^{-38} a 10^{38}	7
double	8	10^{-308} a 10^{308}	15
long double	10	10^{-4932} a 10^{4932}	19

altri tipi di variabili predefiniti

char, bool (può assumere solo i 2 valori true/false)

N.B

- **const** int a = 4; (a non può più essere rassegnata)
- **typedef** float Real; (Real diventa sinonimo di float)

Operatori

+	**addizione**
-	**sottrazione**
*	**moltiplicazione**
/	**divisione**
%	**modulo**
++	**incremento**
--	**decremento**
=	**assegnazione**

assegnazione multipla: posso assegnare lo stesso valore a più variabili

var1 = var2 = var3 = valore;

assegnazione composta: si eseguono i calcoli prima dell'assegnazione:

x += y; è come x = x + y ;

i += 2; è come i = i+2;

N.B. incremento: per incrementare di 1 la variabile z si può scrivere:

z ++ oppure ++ z

cioè mettere l'operatore ++ prima o dopo del nome della variabile. Le due forme non sono sempre equivalenti. La differenza si esplicita quando si valuta una *espressione* che contiene z++ o ++z.
Scrivendo z++, il valore di z viene *prima usato poi incrementato*.
Scrivendo ++z, il valore di z viene *prima incrementato e poi usato*.
Basta tenere presente che l'ordine delle operazioni avviene sempre da sinistra verso destra.

l'operatore ?

L'operatore di assegnazione condizionata **?** ha la seguente sintassi:
espressione_logica ? espr1 : espr2
Se espressione_logica è **vera** restituisce **espr1 altrimenti** restituisce **espr2.**
Si utilizza tale operatore per assegnare, condizionatamente, un valore ad una variabile.

Esempio |x|
valore_assoluto = x>0 ? x : -x;

Tavola riassuntiva

Operatore	Esempio	Risultato
!	!a	(NOT logico) 1 se a è 0, altrimenti 0
<	a < b	1 se a<b, altrimenti 0
<=	a <= b	1 se a<=b, altrimenti 0
>	a > b	1 se a>b, altrimenti 0
>=	a >= b	1 se a>=b, altrimenti 0
==	a == b	1 se a è uguale a b, altrimenti 0
!=	a != b	1 se a non è uguale a b, altrimenti 0
&&	a && b	(AND logico) 1 se a e b sono veri, altrimenti 0: se a è falso, b non viene valutato
||	a || b	(OR logico) 1 se a è vero, (b non è valutato), 1 se b è vero, altrimenti 0

Funzioni di libreria

Richiedono tutte **#include <math.h>**

|x|	fabs(x)
$\sqrt{x}$	sqrt(x)
x^a	pow(x,a)
e^x	exp(x)
ln(x)	log(x)
$\log_{10}(x)$	log10(x)
sen(x)	sin(x)
cos(x)	cos(x)
tg(x)	tan(x)
arcsen(x)	asin(x)
arccos(x)	acos(x)
arctg(x)	atan(x)
senh(x)	sinh(x)
cosh(x)	cosh(x)
tgh(x)	tanh(x)
	ceil(x) restituisce il più piccolo intero >= x
	floor(x) restituisce il più grande intero <= x

Funzioni create dall'utente

In C++ la funzione è il solo strumento per realizzare sottoprogrammi e procedure.
Una funzione può avere dei parametri di ingresso e, se non è definita di tipo **void**, restituire un risultato. Sintassi:

Tipo del risultato ***nome_funzione (****tipo1 inp1, tipo2 inp2, …, tipon inpn****)***

```
{
        ….
        return risultato;
}
```

Le funzioni possono essere definite prima della dichiarazione **main** oppure dopo: nel secondo caso (ed è la prassi) si deve inserire prima del main il ***prototipo*** *(prototype)* della funzione, ossia la sua intestazione.

In una funzione possono esserci più istruzioni ***return.*** Vedi l'esempio sottostante:

```
int match (int a, int b, int c) {
  if (a == b)
        return c;
   else
        return b;
}
```