La scorsa volta abbiamo visto come **una variabile rappresenta** una **zona di memoria di una specifica grandezza dipendente dal tipo di variabile**.
I tipi di variabili si distinguono per due fattori molto importanti:

- La quantità di memoria occupata
- La codifica

Breve parentesi sul significato di codifica

Quando abbiamo dei dati, come già visto, dobbiamo sapere cosa **rappresentano** per poterne **estrapolare l'informazione giusta**.

Per esempio, immaginiamo di avere il numero 36, questo rappresenta il numero *trentasei* perché noi siamo abituati a leggere i numeri in formato decimale.
Ma 36 potrebbe anche significare trentasei migliaia, oppure potrebbe rappresentare il carattere numero 36 della tabella ASCII.

Quindi **è importante sapere come vanno letti** i dati che abbiamo a disposizione.

Parlando in modo specifico, nello spazio di memoria assegnato alla variabile, come sappiamo, sono presenti degli zeri e degli uni, ma poiché esistono diversi modi per rappresentare numeri interi senza segno, interi con segno, numeri con la virgola, caratteri, il computer ha bisogno di sapere cosa rappresentano quegli zeri e uni.
Es:
1001 se letto senza segno rappresenta il numero 9, ma se letto con segno rappresenta -7

I principali tipi di variabile in C

Tipo	*Spazio occupato (per sistemi a 64 bit)*	*Descrizione*
int	4 byte	Un numero intero con un valore compreso tra 2147483647 e -2147483648
long int	8 byte	Un numero intero con un valore compreso tra 9223372036854775807 e -9223372036854775808
short int	2 byte	Un numero intero con un valore compreso tra 32767 e -32768
unsigned int	4 byte	Un numero intero con un valore compreso tra 0 e 4294967295
char	1 byte	Un carattere della tabella ascii da 0 a 255 (che trovate qui) vedremo nella prossima lezione come trattarli
float	4 byte	Un numero con la virgola

double	8 byte	Un numero con la virgola più preciso

Come vedete, **ogni tipo ha la sua dimensione** e **ogni tipo ha la sua codifica** per rappresentare l'informazione a esso assegnata.
Come abbiamo visto la scorsa volta, per **dichiarare una variabile** di un certo tipo non dobbiamo far altro che **scrivere il tipo e la variabile**:
double variabile;
E per assegnarle un valore scriveremo:
variabile = 123.456;
In questo modo abbiamo **creato uno spazio di 8 byte** e ci abbiamo scritto dentro il **valore 123.456 codificato come double**.

E' importante ricordare che le variabili non possono né cambiare tipo, né cambiare la quantità di memoria occupata!

Le espressioni

Quando assegnamo un **valore** ad una variabile **questo è il risultato di un'espressione**.
Un'espressione è tipicamente formata da variabili, valori e operatori.
C'è una frase che è molto importante che ricordiate ogni volta che scrivete espressioni:
"*Ogni espressione e ogni variabile ha un tipo e un valore*"

Nell'esempio precedente, 123.456 è un valore di tipo float, **quando viene assegnato** alla variabile "variabile" questo valore viene "**castato**", ovvero **trasformato** in double, dopodiché viene assegnato.
Questo succede anche se scriviamo
variabile = 1;
In questo esempio 1 **è un valore di tipo** int, esso **viene castato** a double **e poi assegnato**.
E' molto utile tenere a mente che **è possibile forzare il cast** di un valore scrivendo il tipo del valore tra parentesi in questo modo
variabile = (float) 1;
In questo modo 1 **viene castato** a float e successivamente **viene automaticamente castato** a double e assegnato.
Il cast è una cosa molto frequente e soprattutto **automatica**. Per questo **bisogna fare attenzione** poiché un cast da float a int ci fa **perdere l'informazione della virgola**:
float var = 12.34;
var = (int) var;
In queste righe abbiamo assegnato il numero 12.34 a var, dopodiché abbiamo riassegnato var a var stessa, ma prima di farlo abbiamo castato var a int.
Di conseguenza adesso in var sarà salvato il valore 12.0, quindi **abbiamo perso l'informazione dopo la virgola**.
Tenete a mente che il **cast non approssima**, **tronca solo**, quindi **se abbiamo 12.9**, esso **diventerà 12.0**

"(int) var" **è quindi un'espressione**.
Nelle espressioni possiamo inserire anche degli **operatori**, cosa sono gli operatori:
gli operatori sono quelli che conosciamo anche in matematica: + - * /
operatori come questi vengono chiamati **operatori binari**, perché? Perché **prendono in ingresso due valori**, per esempio a+b, b/c
In generale **esistono anche operatori unari**, per esempio in matematica il simbolo ! indica

il fattoriale: !8 = 2*3*4*5*6*7*8, come vedete il fattoriale **prende in ingresso solo un valore**.
In ogni caso, gli unici operatori matematici presenti in C sono **addizione, sottrazione, moltiplicazione e divisione**, più alcuni operatori che operano sui **numeri binari** che non ci interessano.
Vediamo alcuni esempi

```
int a;
int b;
a = 5;
b = a+7;
b = a*b;
```

Ora, in questo modo tratto solo valori interi, ma che succede se divido due numeri interi?

```
int a, b;
float c;
a = 10;
b = 4;
c = a/b;
```

Che succede in questo caso? a e b vengono divisi con la cosiddetta **divisione piana**, ovvero, **senza virgola**: dentro c sarà memorizzato 2.0
Castando invece **almeno uno** dei due numeri a float l'altro verrà **automaticamente castato**

```
c = a/(float)b;
c = (float)a/b;
```

Tenete presente che questo **accade anche** per i numeri "costanti", dopo tutto **anch'essi sono espressioni** a sé e quindi hanno un tipo e un valore!

```
c = 10/3;
c = 10.0/3
c = 10/3.0
c = 10/(float)3
```

Di tutti questi casi, il primo salverà in c il valore 3.0, mentre tutti gli altri salveranno 3.33333... poiché scrivere .0 indica che il **valore è con la virgola**

Ovviamente come in matematica **gli operatori hanno delle precedenze** sugli altri, a+b*c eseguirà **prima** la moltiplicazione e **poi** l'addizione.
Se abbiamo bisogno di **eseguire un'espressione prima di un'altra**, possiamo **in ogni caso** utilizzare le parentesi: (a+b)*c

Se vogliamo possiamo considerare il cast come un operatore unario.

Ricordate che il C **cerca come può** di castare **automaticamente** i valori per **evitare** di eseguire **operazioni tra dati con significati diversi** e, **di solito**, il criterio è quello di castare al **valore che occupa più byte**, ma, **se avete dubbi sul suo comportamento**, o volete evitarlo, potete **sempre** castare, **non vi costa niente e vi fa stare tranquilli**.