
Costanti

Le costanti sono entità che non possono essere modificate durante l'esecuzione del programma, ma mantengono invariato il loro valore per tutta la durata del programma.
Le costanti possono essere:

1. **scritte direttamente nel programma** scrivendo il valore che interessa.
   Ad es. i = 1; f = 137.12; k = 0xFF;
2. **indicate mediante un simbolo tramite una direttiva al preprocessore #define** nel qual caso il preprocessore, sostituendo al simbolo il valore, ci riporta al caso precedente,
   Ad es. #define SIZE 10
   int i = SIZE;
   viene tradotto dal preprocessore in
   int i = 10;
3. **indicate mediante un simbolo** (tramite la keyword **const**) che viene inizializzato al valore che interessa e mantenuto in una locazione di memoria come una variabile, ma che non può essere modificato.
   La dichiarazione di un dato di tipo costante deve essere così:
   const Tipo NomeVariabile = costante;
   oppure
   Tipo const NomeVariabile = costante;
   Ad es. const int i = 10;
   Dichiarare una costante mediante la const impone al compilatore di associare il simbolo ad una certa locazione di memoria separata dal resto dei dati, e di mantenerne in questa locazione il valore assegnato in via di inizializzazione.

Comunque, in tutti i 3 casi, resta sempre il problema di come scrivere un certo valore costante (numerico intero, numerico in virgola mobile, stringa) all'interno in una certa istruzione. Il formato varia al variare del tipo di dato.

Esiste inoltre un caso particolare di uso delle costanti che sarà descritto a proposito delle funzioni e dei loro parametri formali, e che serve a dichiarare che un parametro formale non è modificabile.

Costanti Numeriche Intere

Le costanti numeriche intere: sono espresse da numeri senza componente frazionaria, possono essere preceduti eventualmente da un **segno** (+ o -), e possono essere espresse in notazione decimale (base 10), esadecimale (base 16) o ottale (base 8).

- In notazione decimale il numero è espresso nel modo consueto,
- in notazione esadecimale il numero deve essere preceduta dal segno **0x** ,
- in notazione ottale il numero deve essere preceduta da uno **0** .

Per indicare il tipo di dato con cui rappresentare la costante, si utilizzano dei caratteri da porre immediatamente dopo i caratteri numerici.

- Per default le costanti intere sono considerate int.
- Se la costante intera è troppo grande per essere contenuto in un int, viene considerato long.
- Una costante unsigned int deve essere seguita dal carattere **u** o **U**.
  es: 41234U
- Una costante long deve essere seguita dal carattere **L** o **l**.
  es: 49761234L
- Una costante unsigned long deve essere seguita dai caratteri **UL** o **ul**.

Esempi:

Valore	Tipo	DECIMALE	ESADECIMALE	OTTALE
0	int	0	0x0	00
1	int	1	0x1	01
8	int	8	0x8	010
-8	int	-8	-0x8	-010
17	int	17	0x11	021
-30	int	-30	-0xFD	036
100000	long int	100000**L** 100000**l**	186A0**L** 186A0**l**	-303240**L** -3032040**l**
-100000	long int	-100000**L** -100000**l**	-186A0**L** -186A0**l**	-303240**L** -3032040**l**

Costanti Numeriche in Virgola Mobile

Le costanti numeriche in virgola mobile sono scritte:

- in **rappresentazione decimale** nel modo consueto, ovvero: un eventuale segno, la componente intera, il punto decimale ed eventualmente la componente decimale.

  es: -10.4 37.235**f** 7. .001

- oppure in **formato esponenziale**, cioè nella forma Xe±M con il significato di $X*10^{\pm M}$, dove X è una componente floating point rappresentata come prima indicato, M è un intero

  es: -1.04e+1 0.37235e+2**L** 0.7e+1 1.e-2

Inoltre, il tipo di dato usato per rappresentare la costante sarà:

- il tipo double se non viene specificato un suffisso alla costante.
- il tipo float se invece viene aggiunto un suffisso **f** o **F** ,
- il tipo long double a 16 byte se viene aggiunto un suffisso **l** o **L**.

Costanti di tipo Char

I caratteri, cioè i tipi char (signed e unsigned) servono a rappresentare un insieme di simboli quali i simboli della tastiera a A b B 1 2 ed alcuni caratteri cosiddetti "di controllo" quali Enter, Tab, BackSpace, NewLine.
Lo standard ASCII associa un carattere ai valori da 0 a 127, mentre per i caratteri da 128 a 255 l'associazione varia a seconda del computer.
I caratteri di controllo sono i primi 32 della tabella ASCII, cui seguono i caratteri stampabili, cioè visualizzabili da un comune editor. Qui di seguito un estratto dalla tabella ASCII.

codice ASCII	48	49	50	51	52	53	54	55	56	57
carattere	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'

codice ASCII	65	66	67	68	69	70	71	ecc..		
carattere	'A'	'B'	'C'	'D'	'E'	'F'	'g'	ecc..		

codice ASCII	97	98	99	100	101	102	103	ecc..		
carattere	'a'	'b'	'b'	'd'	'e'	'f'	'g'	ecc..		

Una variabile di tipo carattere quindi memorizza un certo valore, da 0 a 255, che corrisponde ad un certo carattere. In un assegnamento ad una variabile di tipo carattere, potremo assegnare o il codice numerico o il carattere rappresentato da quel codice numerico.

Una costante di tipo carattere quindi potrà essere rappresentata o come valore numerico da 0 a 255, oppure come simbolo corrispondente a quel codice, ad es. ad un certo char potremo assegnare o il carattere '0' o il valore numerico 48.

La **rappresentazione numerica delle costanti di tipo carattere** è fatta mediante la rappresentazione decimale del codice numerico. Ad es assegnando ad un char il valore 97 gli assegnamo il carattere 'a'.

La **rappresentazione simbolica** delle costanti di tipo carattere, **non soffre dei problemi dovuti alla differenze tra le codifiche dei caratteri**, perchè **scrive nel codice** direttamente **il simbolo** che si vuole ottenere, e non una sua rappresentazione numerica.
D'altro canto è necessario delimitare la rappresentazione simbolica di un certo carattere separandola dagli altri elementi del linguaggio C. A questo scopo **la rappresentazione delle costanti di tipo carattere mediante i simboli viene fatta scrivendo il simbolo all'interno di due apici singoli**.

ad es. '**L**' indica il carattere che rappresenta il simbolo L.

Però, poichè non tutti i caratteri sono visualizzabili da un editor (ad es. i caratteri di controllo) deve essere previsto **un modo simbolico per indicare un carattere che non può essere visualizzato**. A tal scopo sono definite le cosiddette "sequenze di escape", ovvero dei simboli stampabili che quando sono precedute dal carattere \ vengono interpretate dal C come un carattere diverso.

Ad es. la sequenza di escape '**\n**' indica il carattere **new line** (1 carattere in Unix, due caratteri in DOS) cioè l'andata a capo di una riga di testo. Le sequenze di escape vanno racchiuse tra apici singoli come i simboli dei caratteri.

Le sequenze di escape vengono introdotte anche per poter indicare simbolicamente alcuni caratteri particolari (' ") che hanno un significato speciale nel linguaggio C

Le principali sequenze di escape disponibili in C sono le seguenti:

\a	suono
\b	BackSpace
\n	New Line, andata a capo
\r	carriage return
\t	il Tab orrizzontale
\\	il BackSlash, che è il delimitatore delle sequenze di escape
\'	il carattere apice singolo ' che in C è delimitatore di carattere
\"	il carattere doppio apice " che in C è delimitatore di stringa

Notare l'analogia tra l'uso delle sequenze di escape \\ \' \" in C e l'uso della coppia di apici singoli " dentro la write per stampare un solo apice singolo '.

Infine è possibile rappresentare i caratteri ancora in forma numerica mediante rappresentazione ottale o esadecimale, scrivendo all'interno della coppia di apici singoli:
- un BackSlash seguito dalla rappresentazione ottale del numero, oppure
- un BackSlash seguito da una x seguita dalla rappresentazione esadecimale.

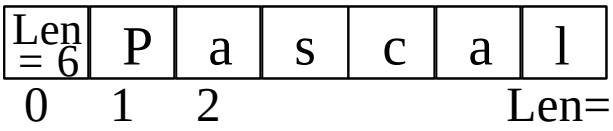
rappr. ottale	'\OOO'	fino a tre caratteri ottali (0:7), (max \377)
rappr. esadecimale	'\xhh'	fino a due (hh) caratteri esadec. (0:9,A:F)

es, sono equivalenti i seguenti assegnamenti, nella stessa colonna:
char ch;

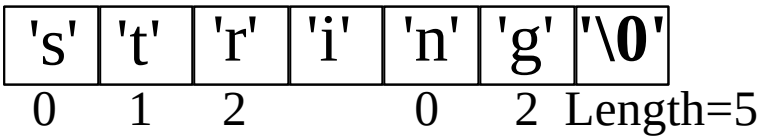
ch='A';	ch=' " ';	ch=' \' ';	ch='\\';
ch=65;	ch=34;	ch=39;	ch=92;
ch='\x41';	ch='\x22';	ch='\x27';	ch='\x5C';
ch='\101';	ch='\42';	ch='\47';	ch='\134';
_____	ch='\042';	ch='\047';	**ch='\';/*errore*/**
	ch='\'";	ch='\047';	
non ammesso	**ch='\0042';**	**ch=' ' ; /*errore*/**	
ch='\400';	**errore,4 ottali**		

Costanti di tipo Stringa

In Pascal, la stringa è un dato primitivo, formato da n caratteri, costituito da un vettore di n+1 posizioni, in cui in prima posizione è contenuto un contatore che indica di quanti caratteri è composta la stringa.



In C invece la stringa vera e propria non esiste, si usa come stringa un vettore di n+1 char , in cui al carattere in ultima posizione è assegnato il valore 0, ovvero il carattere nullo '\0'. Questo carattere particolare rappresenta il delimitatore finale della stringa, ovvero l'indicazione che la stringa è finita.



Quindi, una stringa in C internamente è rappresentata da un vettore di caratteri terminati da un carattere nullo. La lunghezza è nota solo dopo avere trovato il carattere 0 in fondo.

Una costante di tipo stringa, ovvero una costante di tipo vettore di caratteri viene scritta come una **sequenza di caratteri racchiusa tra 2 doppi apici, che non fanno parte della stringa**.

Ciascun carattere nella stringa può essere rappresentato simbolicamente o per mezzo di una sequenza di escape o in forma esadecimale o ottale.

Esempi di costanti stringa sono:

"questA stringa è giusta" 'è' e' un caratt. non ASCII

"quest\x41 stringa e' giusta" ' ' ' è delimitatore di carattere, non di stringa, quindi non c'e' confusione

"quest\x41 stringa e\47 giusta" ' ' ' scritto in ottale con 2 cifre(occhio)

"quest\101 stringa e\047 giusta" ' ' ' scritto in ottale a 3 cifre

"questA stringa " è sbagliata" ERRORE, "" è delimitatore di stringa

"questA stringa \" è stata corretta"

"aa12bb" è uguale a "aa**\061**2bb" ma è diverso da "aa**\61**2bb"

NB: la const a destra dà errore in compilazione,"numeric constant too large"

Type Casting

Molto spesso il risultato di un'operazione dipende dal tipo di dato che è coinvolto nell'operazione. Ad es. la divisione tra interi ha un risultato diverso dalla divisione tra floating point, anche partendo dagli stessi dati iniziali. 5.0/2.0=2.5 5/2=2

Questo perchè a seconda del tipo di dati coinvolti nelle operazioni, le operazioni sono svolte in modo diverso, o utilizzando registri o porzioni di registri diversi.

Una **caratteristica peculiare del C** è che **permette** durante **l'esecuzione di un'operazione**, o **durante il passaggio di parametri ad una funzione all'atto della chiamata**, **di modificare il tipo del o dei dati coinvolti nell'operazione**.

Ciò non significa affatto che la variabile (o la costante, o il risultato di un'espressione) interessata **cambia le sue caratteristiche** (dimensioni, proprietà, ecc..) di tipo.

Significa invece che se **la variabile** deve essere utilizzata in dei calcoli, **viene copiata in un registro sufficentemente grande** per contenere il **nuovo tipo di dato** e per **svolgere le operazioni di conversione**, ed il **valore del registro caricato viene convertito** secondo le caratteristiche del nuovo tipo di dato.

Solo **dopo questa conversione, l'operazione viene effettuata, secondo le regole proprie del nuovo tipo di dato**, ottenendo quindi come risultato un valore coerente con il nuovo tipo.

La conversione di tipo **type-casting** in qualche caso viene effettuata implicitamente dal compilatore C, ma può anche essere forzata dal programmatore mediante un operatore unario detto **cast**, che opera in questa maniera.

(nome_nuovo_tipo) espressione

dove "nome_nuovo_tipo" è un tipo di dato primitivo o definito dall'utente, ed "espressione" può essere una variabile, una costante o un'espressione complessa. es. (double) i;

"Espressione" viene risolta fino ad arrivare ad un risultato (di un suo certo tipo), poi il risultato viene convertito nel tipo "nome_nuovo_tipo" mediante opportune operazioni più o meno complesse a seconda della conversione.
A questo punto abbiamo come risultato della conversione un dato di tipo "nome_nuovo tipo" con un certo valore, che può essere utilizzato secondo le regole definite per "nome_nuovo_tipo".

Vediamo subito un esempio di cosa succede:

```
int    i=5,    j=2;
double f;
f = i / j;
/* f ora vale 2 */
```

```
int i=5,    j=2;
double f;
f = (double)i / (double) j;
/* f ora vale 2.5 */
```

Un cast quindi equivale ad assegnare l''espressione'' ad una variabile del nuovo tipo specificato, che viene poi utilizzata al posto dell'intera espressione.

L'esempio riportato mostra un caso in cui il type cast modifica fortemente il risultato di un'operazione, che comunque avrebbe potuto essere effettuata, pur con risultati diversi, senza type cast.

Il **casting viene** però **utilizzato spesso anche nel passaggio di parametri a funzioni,** qualora il programmatore abbia necessità di passare alla funzione chiamata un parametro formalmente diverso da quello richiesto dalla funzione. Ad es qualora si debba passare alla funzione un puntatore ad una struttura, di tipo diverso da quella che la funzione si aspetta.
Il compilatore dovrebbe segnalare l'errore. Mediante un type cast del parametro passato all'atto della chiamata (di cui il programmatore si deve assumere la responsabilit relativamente alla correttezza) si può "**imbrogliare**" il compilatore, convincendolo della correttezza formale dell'operazione.

Abbiamo finora parlato di type-casting forzato, ovvero imposto dall'utente.
In realtà il **C effettua automaticamente delle conversioni implicite di tipo**, in particolare quando effettua operazioni matematiche tra operandi di tipo diverso. **Il casting viene effettuato automaticamente dal compilatore C quando due operandi di un'operazione binaria sono tra loro diversi. In tal caso l'operando di tipo più piccolo viene convertito nel tipo più grande, senza perdita di informazioni.**
Quindi, dati due operandi di tipo diverso, il cast automatico si ha secondo queste regole:

Tipo 1° operando	Tipo 2° operando	Tipo Risultato
long double	double	long double
double	float	double
float	(long) int	float
double	(long) int	double
int	char , short	int
long int	(short) int	long int

Attenzione, il casting automatico può dare delle false sicurezze. Riguardare l'esempio precedente per verificare per quale motivo si ha perdita di informazioni (risposta: dipende dall'ordine di esecuzione delle operazioni. Prima viene fatta la divisione, sono due interi quindi nessuna conversione e c'è perdita di informazioni, e solo in un secondo tempo c'e' l'assegnamento al float).

```
int i=5, j=2;      double f;
f = (double) (i /j);          f diventa 2.0
```

N.B. come abbiamo già visto, anche nell'assegnamento il C effettua conversioni di tipo automatiche, convertendo il valore del lato destro nel tipo del lato sinistro, eventualmente perdendo informazioni quando si passa da un tipo ad un tipo più piccolo.

Ora che abbiamo considerato il problema della conversione di tipo, vediamo quali sono e cosa fanno alcuni operatori del C