



Tipi di dato semplici

Informatica B

Tipi di dati

- ❑ In C esistono diversi tipi di dato **built-in**, tra cui
 - ▶ **int**: numeri interi
 - ▶ **float**: numeri con virgola (singola precisione)
 - ▶ **double** : numeri con virgola (doppia precisione)
 - ▶ **char**: caratteri (0-255)
- ❑ Inoltre il C fornisce anche la possibilità di definire dei nuovi tipi di dato

Il tipo intero

- ❑ Consente di rappresentare numeri interi
- ❑ Il tipo base per lavorare con numeri interi in C è l'`int` (abbreviazione di `integer`)
- ❑ In C esistono inoltre diverse varianti:
 - ▶ `short int`: per numeri interi di piccole dimensioni
 - ▶ `long int`: adatti a numeri interi di grandi dimensioni
 - ▶ `unsigned`: per indicare che si utilizzerà la variabile solo per numeri positivi
- ❑ La quantità di memoria usata per ciascun tipo non è definita dal C ma dipende dal compilatore e dalla macchina
- ❑ È richiesto solo che sia vera la seguente relazione:
$$\text{mem}(\text{short int}) \leq \text{mem}(\text{int}) \leq \text{mem}(\text{long int})$$
- ❑ Spesso la dimensione dell'`int` corrisponde alla parola di memoria, mentre per lo `short` si usano 2 byte e per il `long` 4 byte
- ❑ I tipi `unsigned` permettono di rappresentare numeri più grandi, non dovendo rappresentare i numeri negativi

Il tipo intero: dichiarazioni

```
int a; /* Dichiarazione */  
a = 0; /* Inizializzazione */
```

```
int a = 0; /* Dichiarazione ed inizializzazione */
```

```
int a = 2147483647; /* In sistemi a 32 bit gli int permettono di rappresentare */  
int a = -2147483648; /* numeri compresi fra  $-2^{31}$  (-2147483648) a  $2^{31} - 1$  (2147483647) */
```

```
unsigned int a = 4294967295; /* In sistemi a 32 bit gli unsigned int permettono di */  
unsigned int a = -1; /* rappresentare numeri compresi fra 0 e  $2^{32} - 1$  (4294967295) */
```

```
short int a = 32767; /* In sistemi a 32 bit gli unsigned int permettono di rappresentare */  
short int a = -32768; /* numeri compresi fra  $-2^{15}$  (-32768) e  $2^{15} - 1$  (32767) */
```

```
short unsigned int a = 65535; /* In sistemi a 32 bit gli unsigned short int permettono di */  
short unsigned int a = -1; /* rappresentare numeri compresi fra 0 e  $2^{16} - 1$  (65535) */
```

```
long int a = 4294967295; /* In sistemi a 32 bit i long int sono equivalenti agli int */
```

Il tipo intero: operazioni aritmetiche

```
int a,b,c;  
b=5;  
c=3;
```

Le parentesi consentono di cambiare l'ordine di esecuzione

```
a = b+c+10; /* a <-- 18 */  
a = (b+3)*c; /* a <-- 24 */  
a = 4+b-c; /* a <-- 6 */  
a = b/c; /* a <-- 1 */  
a = b%c; /* a <-- 2 */
```

Divisione intera

```
a+=3; /* a=a+3 */  
a-=3; /* a=a-3 */  
a*=3; /* a=a*3 */  
a/=3; /* a=a/3 */
```

Resto divisione intera

```
a++; /* a=a+1 */  
a--; /* a=a-1 */
```

Il tipo intero: lettura e scrittura

```
int i;  
unsigned int u;  
short s;  
unsigned short us;  
long int l;  
unsigned long int ul;
```

```
printf("Inserire un int: ");  
scanf("%d",&i);  
printf("Inserire un unsigned int: ");  
scanf("%u",&u);  
printf("Stampo %d come unsigned int: %u\n",-1,-1);  
printf("Inserire uno short: ");  
scanf("%d",&s);  
printf("Inserire un unsigned short: ");  
scanf("%u",&us);  
printf("Inserire un long: ");  
scanf("%ld",&l);  
printf("Inserire un unsigned long: ");  
scanf("%lu",&ul);
```

Stampo -1 come unsigned int:
4294967295

Il tipo reale

- ❑ I tipi per lavorare con numeri reali in C sono il **float** (abbreviazione di **floating point**) e il **double**
- ❑ Si differenziano per la loro precisione: il **double** permette infatti di rappresentare i numeri reali in un intervallo più grande e con maggiore precisione
- ❑ La quantità di memoria usata per ciascuna variante non è definita dal C ma dipende dal compilatore e dalla macchina
- ❑ È richiesto solo che sia vera la seguente relazione:
 $\text{mem}(\text{float}) \leq \text{mem}(\text{double}) \leq \text{mem}(\text{long double})$
- ❑ Tuttavia nella maggior parte dei casi, il **float** occupa 4 byte mentre il **double** 8 byte

Il tipo reale: assegnamenti

```
float f1 = 1.045;
```

```
float f2 = .855; /* --> a = 0.855 */
```

```
float f3 = 4.5567e3; /* --> a = 4556.7 */
```

```
float f4 = 4.53e-2; /* --> a = 0.0453 */
```

```
double d1 = .0005;
```

```
double d2 = -4.3e50; /* double adatti per numeri molto grandi */
```

```
double d3 = 4.2e-78; /* e molto piccoli (elevata precisione) */
```


Il tipo reale: operazioni aritmetiche

```
float a,b,c;
```

```
b=5.3;
```

```
c=3.2;
```

```
a = b+c+.4; /* a <-- 8.9 */
```

```
a = b*c; /* a <-- 5.3*3.2 = 16.96 */
```

```
a = 4+b-c; /* a <-- 6.1 */
```

```
a = (b-.3)/(c+0.8); /* a <-- 5/4 = 1.25*/
```

```
a = 5/4; /* a <-- 1 - Eseque una divisione intera!!!*/
```

Il tipo reale: lettura e scrittura

```
float f;  
double d;  
printf ("Inserire un numero float: ");  
scanf("%f",&f);  
printf("Il numero letto e' %f\n", f);  
printf ("Inserire un numero float: ");  
scanf("%f",&f);  
printf("Il numero letto e' %f\n", f);  
printf("Lo stesso numero con spec g e' %g\n", f);  
printf ("Inserire un numero double: ");  
scanf("%lf",&d);  
f=d;  
printf("Il numero letto e' %lf\n", d);  
printf("Il numero letto in versione float %f\n", f);
```


Esempio

- ❑ Scrivere un semplice programma che converte una temperatura da gradi Fahrenheit a gradi Celsius
$$[^{\circ}\text{C}] = ([^{\circ}\text{F}] - 32) \cdot 5/9$$

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float tempF, tempC;
```

```
    printf("Inserire temperatura in gradi Fahrenheit da convertire: ");
```

```
    scanf("%f",&tempF);
```

```
    tempC = (tempF-32) * (5.0/9.0); /* Non usare 5/9 !!!!*/
```

```
    printf ("Temperatura convertita in Celsius %f\n",tempC);
```

```
    return 0;
```

```
}
```

Tipo carattere

- ❑ Il C mette a disposizione il tipo `char` che può contenere un carattere
- ❑ Il `char` viene rappresentato solitamente con 1 Byte e contiene la codifica numerica del carattere: un valore nell'intervallo $[0,255]$
- ❑ Caratteri speciali
 - ▶ `'\n'` a capo
 - ▶ `'\t'` tab
 - ▶ `'\r'` carriage return
 - ▶ `'\b'` backspace

Tipo carattere: assegnamenti e operazioni

```
char a;
```

```
char b, c = 'q'; /* Le costanti di tipo carattere si indicano con ' */
```

```
a = "q"; /* NO: "q" è una stringa, anche se di un solo carattere */
```

```
a = '\n'; /* OK: \n è un carattere a tutti gli effetti */
```

```
c = 'ps'; /* NO: 'ps' non è un carattere valido */
```

```
a = 75; /* Che cosa succede? */
```

```
a = 'c' + 1; /* a <-- 'd' */
```

```
a = 'c' - 1; /* a <-- 'b' */
```

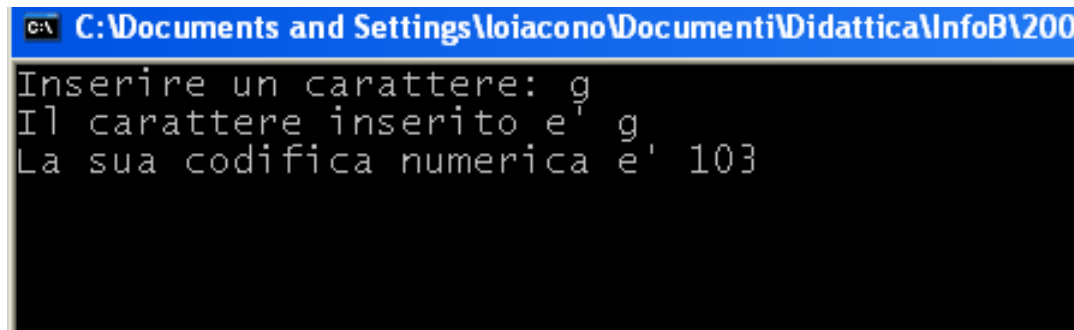
```
a = 20;
```

```
a *= 4;
```

```
a -= 10; /* a <-- 70 che corrisponde al carattere 'F' */
```

Tipo carattere: lettura scrittura

```
char c;  
printf("Inserire un carattere: ");  
scanf("%c", &c);  
printf("Il carattere inserito e' %c\n",c);  
printf("La sua codifica numerica e' %d\n",c);
```



```
C:\Documents and Settings\loiacono\Documenti\Didattica\InfoB\200  
Inserire un carattere: g  
Il carattere inserito e' g  
La sua codifica numerica e' 103
```

Tipo carattere: esempio 1

- ❑ Scrivere un programma per “tradurre” la codifica numerica in carattere

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    char c;
```

```
    printf("Inserire la codifica numerica: ");
```

```
    scanf("%d", &c);
```

```
    printf("La traduzione e' %c\n",c);
```

```
    return 0;
```

```
}
```


Tipo carattere: esempio 2

- ❑ Scrivere un programma per trasformare le maiuscole in minuscole

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    char c;
```

```
    printf("Inserire il carattere maiuscolo: ");
```

```
    scanf("%c", &c);
```

```
    printf("La traduzione e' %c\n",c+32);
```

```
    return 0;
```

```
}
```

Ch	N
'A'	65
...	...
'Z'	90
...	...
'a'	97
...	...
'z'	122

Tipi enumerativi

- ❑ Consente di elencare una sequenza di valori simbolici che la variabile può assumere
- ❑ I valori vengono poi codificati come interi (a partire da 0)
- ❑ Aumenta la leggibilità del programma
- ❑ Sintassi: `enum {val1,val2,...,valN} <nome_variabile>`
- ❑ Esempio:

```
enum {falso,vero} condizione, condizione2;
```

```
condizione = falso;  
condizione2 = vero;
```

```
printf("condizione = %d\n",condizione); /*Stampa 0 */  
printf("condizione2 = %d\n",condizione2); /*Stampa 1 */
```

```
condizione=1; /* assegnamento valido ma... */  
condizione2=20; /* assegnamento valido ma... */
```

Ridefinizione di tipo

- ❑ Consente di ridefinire un tipo semplice con un nuovo nome
- ❑ Sintassi: `typedef <tipo> <nuovo_tipo>`
- ❑ Flessibilità e leggibilità
- ❑ Esempi:

```
typedef int colore;  
colore coloreMacchina;  
coloreMacchina = 5;
```

```
typedef enum {lun,mar,merc,gio,ven,sab,dom} giorno;  
giorno oggi;  
oggi = gio;
```

Costanti

- ❑ Possono essere definite in due modi
 - ▶ `const <tipo> <nome> = val;`
 - ▶ `#define <nome> <valore>`
- ❑ Flessibilità e leggibilità programma
- ❑ Esempi

```
#include<stdio.h>
```

```
#define vero 1
```

```
int main()
```

```
{
```

```
    const int falso = 0;
```

```
    return 0;
```

```
}
```

Conversioni di tipo

- ❑ Il C è un linguaggio con forte tipizzazione: occorre dichiarare il tipo di ogni variabile
- ❑ Questo consente una verifica maggiore durante la fase di compilazione
- ❑ Il C consente di eseguire espressioni ed assegnamenti solo se le variabili coinvolte hanno tipi compatibili
- ❑ Nel caso delle espressioni aritmetiche, se gli operandi sono di tipo diverso, il tipo inferiore viene convertito temporaneamente al tipo superiore, secondo la regola:

`int < long < unsigned < unsigned long < float < double`