Titolo unità didattica: Introduzione al linguaggio C [03]

Titolo modulo : Variabili e tipi in C [03-C]

Sviluppo di semplici programmi C

Argomenti trattati:

- ✓ tipi di dati scalari in C
- ✓ variabili e costanti in C
- ✓ operazione di assegnazione in C
- ✓ operatori aritmetici ed espressioni in C

Prerequisiti richiesti: AP-02-*-T, AP-03-02-C

tipo di dato:

- un insieme di **valori** e un insieme di **operazioni** che si possono essere effettuare su tali valori
- un **criterio di rappresentazione** in memoria, che stabilisce le modalità in cui i valori del tipo sono memorizzati nelle **celle** (voci, locazioni,…) della memoria

tipi di dati scalari in C
(tipi semplici built-in)

- ✓ tipo **intero**
- ✓ tipo **reale**
- ✓ tipo **carattere**

in C è possibile definire **nuovi tipi** di dati scalari
(tipi **user-defined**)

tipo **intero** in C

```
int
```

```
short
long
unsigned short
unsigned int
unsigned long
```

tipo **intero** in C	insiemi dei valori (pc classici, celle di 32 bit)
`int`	-2˙147˙483˙648, +2˙147˙483˙647
`short`	-32˙768, +32˙767
`long`	come `int`
`unsigned short`	0, +65˙535
`unsigned int`	0, +4˙294˙967˙295
`unsigned long`	come `int`

tipo **intero** in C

specificazione di un valore del tipo

numeri interi positivi

[+]ddddddddddd

numeri interi negativi

-ddddddddddd

il numero delle cifre (**d**) dipende dal particolare tipo C
(dal numero di bit per la rappresentazione in memoria)

tipo **intero** in C

il comando C

```
sizeof(tipo)
```

restituisce il **numero di byte** necessario per la rappresentazione di un valore del tipo

```
sizeof(int)
sizeof(unsigned short)
sizeof(long)
```

tipo **reale** in C

`float`

singola precisione,
8 cifre significative

`double`

doppia precisione,
16 cifre significative

`long double`

tipo **reale** in C

specificazione di un valore del tipo

valori **`float`** positivi

`[+]dddd.ddddF`

`[+]dddd.ddddE[±]eeF`

valori **`float`** negativi

`-dddd.ddddF`

`-dddd.ddddE[±]eeF`

il numero delle cifre significative (**`d`**) è al più 8

il numero delle cifre dell'esponente (**`e`**) è al più 2

tipo **reale** in C

specificazione di un valore del tipo

valori **double** positivi

[+]ddddddddd.ddddddddd

[+]ddddddddd.dddddddddE[±]eee

valori **double** negativi

-ddddddddd.ddddddddd

-ddddddddd.dddddddddE[±]eee

il numero delle cifre significative (**d**) è al più 16

il numero delle cifre dell'esponente (**e**) è al più 3

tipo **reale** in C

il comando C

```
sizeof(tipo)
```

restituisce il **numero di byte** necessario per la rappresentazione di un valore del tipo

```
sizeof(float)
sizeof(double)
sizeof(long double)
```

tipo **carattere** in C

`char`

un solo carattere dell'alfabeto esteso

codifica ASCII → 1 carattere → 1 byte

alfabeto esteso: 128 caratteri

000	001	002	003	004	005	006	007	008
009	010	011	012	013	014	015	016	017
018	019	020	021	022	023	024	025	026
027	028	029	030	031	032	033 !	034 "	035 #
036 $	037 %	038 &	039 '	040 (	041)	042 *	043 +	044 ,
045 -	046 .	047 /	048 0	049 1	050 2	051 3	052 4	053 5
054 6	055 7	056 8	057 9	058 :	059 ;	060 <	061 =	062 >
063 ?	064 @	065 A	066 B	067 C	068 D	069 E	070 F	071 G
072 H	073 I	074 J	075 K	076 L	077 M	078 N	079 O	080 P
081 Q	082 R	083 S	084 T	085 U	086 V	087 W	088 X	089 Y
090 Z	091 [	092 \	093]	094 ^	095 _	096 `	097 a	098 b
099 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k
108 l	109 m	110 n	111 o	112 p	113 q	114 r	115 s	116 t
117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }
126 ~	127							

tabella ASCII

tipo **carattere** in C

specificazione di un valore del tipo

un solo carattere dell'**alfabeto esteso**

'k'

l'alfabeto esteso contiene caratteri minuscoli, maiuscoli, simboli speciali, etc…

'K'	**'a'**	**'A'**
'5'	**' '**	**'%'**
'\''	**'\n'**	**'\\'**

tipi **scalari** in C:
spazio di memoria (in byte) per la rappresentazione dei valori

	`sizeof`
`int`	4 byte
`float`	4 byte
`double`	8 byte
`char`	1 byte

pc classici, celle di 32 bit

il tipo **logico** in C **non esiste**

in sostituzione, si usa il tipo **`int`**, con la convenzione che

falso ⟺ **0**

vero ⟺ **1**

qualunque valore **non nullo** è interpretato come **vero**

variabili in C

una variabile C è caratterizzata dal

- ✓ **nome** (identificatore)
- ✓ **valore** associato
- ✓ **tipo**
- ✓ **indirizzo** della cella di memoria a partire dal quale è memorizzato il valore

attenzione!

nome, tipo e indirizzo non possono essere modificati

variabili in C

- il **nome** di una variabile è un identificatore C, cioè una sequenza di lettere e cifre
  - non può iniziare con un numero
  - è formato al più da 31 lettere
  - lettere maiuscole e minuscole sono considerate distinte
- a ogni variabile è associata una **cella di memoria** o più celle consecutive, a seconda del suo tipo
- l'**indirizzo** di una variabile è quello della prima cella

variabili in C

metafora della scatola etichettata in uno scaffale

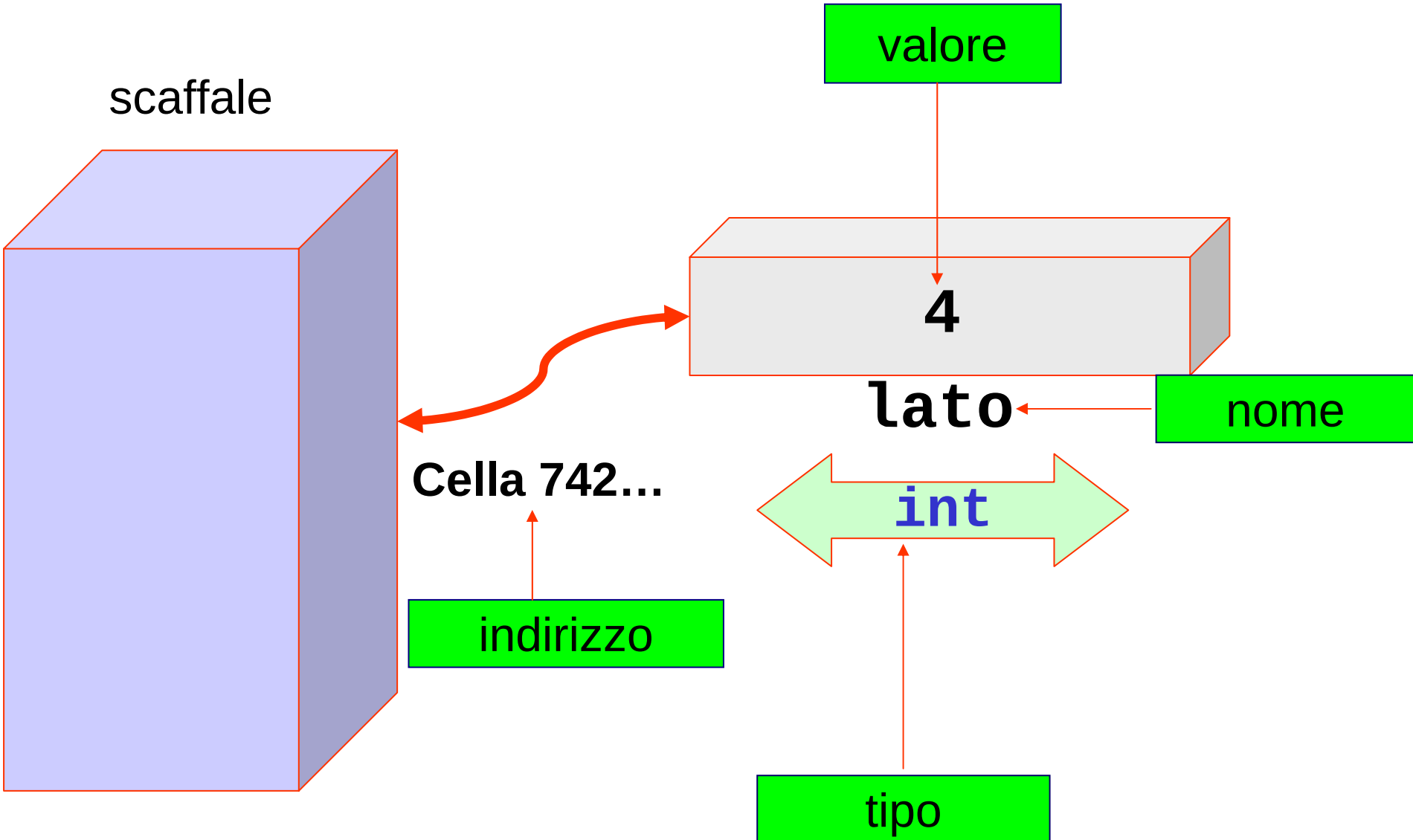
nome ⇨ etichetta

valore ⇨ contenuto della scatola

tipo ⇨ contenuti (e capienza) possibili della scatola

indirizzo ⇨ posizione della scatola nello scaffale

variabili in C



dichiarazione di variabili in C

```
<tipo> <variabili>;
```

Esempio:

```
int eta_anni;
float raggio, circonferenza;
double lato;
char lettera_alfabeto,simbolo;
```

```
var nome_proprio,cognome: string
```

?

dichiarazione di variabili in C

```
<tipo> <variabili>;
```

Esempio:

```
unsigned int eta;
long double velocita_luce;
short indice_riga;
long fattoriale;
int p;
```

costanti in C

una costante è un’associazione non modificabile che associa in modo permanente un valore a un identificatore

```
const float pi_greco = 3.1415926F;
const double pi = 3.14159265258416;
const int n = 100;
```

assegnazione in C

```
<variabile> = <espressione>;
```

simbolo di assegnazione

Esempio:

```
int eta_anni,p;
float raggio;
double lato;
char lettera_alfabeto;
```

```
eta_anni = 27;
raggio = 59.4F;
lato = 3.12;
lettera_alfabeto = ‘g’;
p = 1;
```

assegnazione in C

```
<variabile> = <espressione>;
```

```
var cognome: string
cognome := "Rossi"
```

```
char cognome;
cognome = 'Rossi';
cognome = "Rossi";
```

le stringhe in C sono considerate **valori strutturati**
(verranno trattate nell'ambito delle **strutture dati**)

dichiarazione/inizializzazione in C

```
<tipo> <variabile> = <valore>;
```

Esempio:

```
int eta_anni = 27;
float raggio = 59.4F;
double lato = 3.12;
char lettera_alfabeto = ‘g’;
int p = 1;
```

assegnazione in C

```
<variabile> = <espressione>;
```

operatori aritmetici `+,-,*,/,%`

Esempio:

```
eta_anni = (27-7)/2;
eta_anni = (28-7)/2;
eta_anni = 27-7/2;
raggio = 2.0F*15.4F;
lato = (3.0-0.1)/(3.-1E-1);
eta_anni = 27%5
```

```
10
10
24
30.8
1.0
2
```

operatori aritmetici in C

`+`	addizione (per tutti i tipi)
`-`	sottrazione (per tutti i tipi)
`*`	moltiplicazione (per tutti i tipi)
`/`	divisione (`float`, `double`, `long double`)
`/`	divisione intera (per i tipi interi)
`%`	modulo (per i tipi interi), resto divisione intera

la **divisione intera** (tra un **dividendo di tipo intero** e un **divisore di tipo intero**) fornisce come risultato la **parte intera del quoziente**

`(28-7)/2`	`10`	`1/2`	`0`	`1.0/2.0`	`0.5`

operatori aritmetici in C

`+`	addizione (per tutti i tipi)
`-`	sottrazione (per tutti i tipi)
`*`	moltiplicazione (per tutti i tipi)
`/`	divisione (`float`, `double`, `long double`)
`/`	divisione intera (per i tipi interi)
`%`	modulo (per i tipi interi), resto divisione intera

il **modulo** (tra un **dividendo di tipo intero** e un **divisore di tipo intero**) fornisce come risultato il **resto** della **divisione intera**

`23%7`	`2`	`7%7`	`0`	`1%2`	`1`	`10%2`	`0`	`11%2`	`1`

operatori aritmetici in C

regole di precedenza:

moltiplicazione, divisione e modulo vengono eseguite **prima** di addizione e sottrazione

```
27-8/2
```
23

```
6.0/3.0-1.0
```
1.0

```
16%3+2
```
3

$$\frac{3\cdot 7-1}{4\cdot 2+2}$$

```
3*7-1/4*2+2
```

operatori aritmetici in C

regole di precedenza:

per superare le regole di precedenza è necessario usare le parentesi (tonde)

$$\frac{3 \cdot 7 - 1}{4 \cdot 2 + 2}$$

```
(3*7-1)/(4*2+2)
```

operatori aritmetici in C

regole di precedenza:

se gli operatori hanno la stessa precedenza, allora l'espressione viene valutata da **sinistra verso destra**

```
59*100/60
```
```
98
```
valore **`int`**

```
59.0*100.0/60.0
```
```
98.333333333333
```
valore **`double`**

```
59.0F*100.0F/60.0F
```
```
98.333333
```
valore **`float`**

programma C per il calcolo della **circonferenza** di un cerchio, fissato il suo raggio

commento

```
#include <stdio.h>
/* calcolo circonferenza di un cerchio */
void main ()
{
 const float pi_greco = 3.1415926F;
 float raggio, circon;

 raggio = 2.0F;
 circon = 2.0F * pi_greco * raggio;
 printf ("circonferenza=%f\n",circon);
}
```

visualizzazione su schermo

```
circonferenza=12.566370
Press any key to continue_
```

programma C per il calcolo della **circonferenza** di un cerchio, fissato il suo raggio

```c
#include <stdio.h>
/* calcolo circonferenza di un cerchio */
/* versione doppia precisione */
void main ()
{
 const double pi_greco = 3.14159265258416;
 double raggio, circon;

 raggio = 2.0; /* il Raggio e' fissato */
 circon = 2.0 * pi_greco * raggio;
 printf ("circonferenza=%23.14lf\n",circon);
}
```

parte **dichiarativa**

parte **esecutiva**

```
circonferenza=12.56637061033664
Press any key to continue_
```