



Università degli Studi di Cagliari
Corso di Laurea in Ingegneria Biomedica

ELEMENTI DI INFORMATICA

https://www.unica.it/unica/page/it/gianluca_marcialis

A.A. 2021/2022

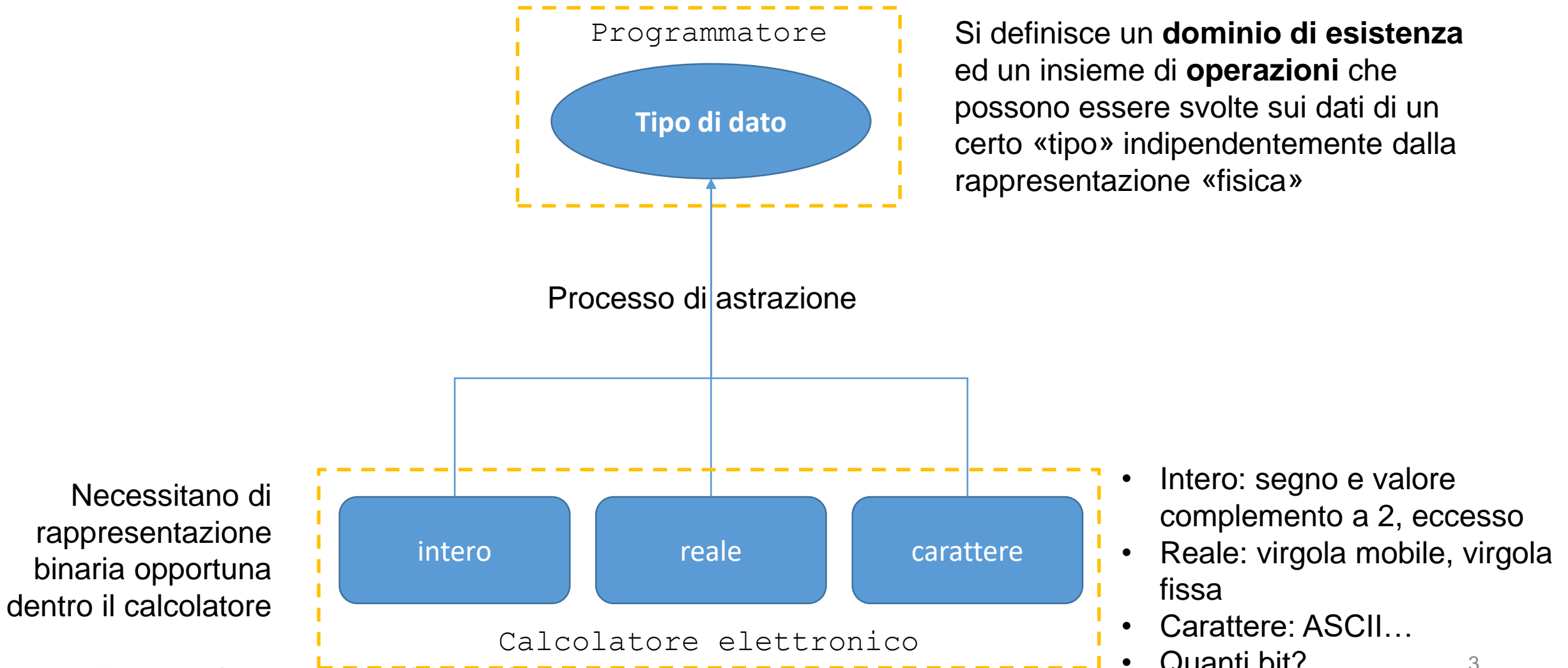
Docente: **Gian Luca Marcialis**

LINGUAGGIO C
Tipi di dati semplici

Sommario

- Classificazione
- Tipi semplici predefiniti
- Definizione di nuovi tipi
- Tipi strutturati

Perché i «tipi di dato»



Classificazione dei tipi di dati

- Tipi semplici
 - L'uso di una variabile numerica è naturale
 - Es. una velocità, una temperatura
- Tipi strutturati
 - Indicano “raggruppamenti” di tipi semplici e strutturati
 - Es. Data: giorno.mese.anno
- In C esistono tipi semplici predefiniti
- Ma l'utente ha la libertà di definirne nuovi, sia semplici, sia strutturati

Tipi semplici predefiniti

Tipi integral		
char	signed char	unsigned char
short	int	long
unsigned short	unsigned	unsigned long
Tipi floating		
float	double	long double
Tipi arithmetic:	Tipi integral + floating	

Il tipo `int`

- Il tipo `int` in C è solo un'approssimazione del corrispondente tipo matematico “intero”
 - Definito su un supporto $(-\infty, \dots, -1, 0, 1, \dots, +\infty)$ attraverso le operazioni di somma, sottrazione, moltiplicazione, divisione
- Infatti ricordiamo che ciascun valore è rappresentato da un insieme di bit, il cui numero è limitato dalle risorse disponibili nel calcolatore
 - Quindi il supporto su cui definire le operazioni è $[-\text{MAXINT}, \dots, -1, 0, 1, \dots, \text{MAXINT}]$, con relativa possibilità di overflow e underflow
 - Tipicamente un `int` è valore intero con segno rappresentabile con 32 bit

Esercizio

- Scrivere un programma in C che, letti due numeri interi da tastiera, stampi a video la loro somma, sottrazione e moltiplicazione, in un'unica riga. Ciascun risultato sia separato da uno spazio.

Varianti del tipo `int`

- `short int`
 - Rappresenta un intero con la metà dei bit di un `int`
- `long int`
 - Usa il doppio dei bit di un `int`
- `unsigned int`
 - Non prevede il segno, permettendo così una rappresentazione più ampia del range non negativo
- `unsigned`, `short` e `long` possono combinarsi
 - Es. `unsigned short int`

Esercizio

- Scrivere un programma C che stampa a video il quoziente ed il resto della divisione di due numeri **interi** X e Y introdotti da tastiera

- Soluzione

```
/*Programma che stampa a video il risultato della divisione di
due numeri introdotti da tastiera*/
#include <stdio.h>
int main()
{
    int x, y, q, r;
    scanf("%d %d",&x,&y);
    q = x/y;
    r = x % y; /* l'operatore % da il resto della divisione*/
    printf("%d / %d = %d con resto %d",x,y,q,r);
    return 0;
}
```

I valori interi visti come «booleani»

- Un variabile intera può essere anche interpretata come booleana secondo questa regola:
 - Qualsiasi valore diverso da zero è True (1)
 - Qualsiasi valore pari a zero è False (0, 0.0, NULL)
- Operazioni su booleani
 - AND, OR, NOT: `&&` `||` `!`
- Come si tradurrà in C l'espressione booleana:
StudiaBiomedica · (*FrequentaPrimoAnno* + *EsamePassato*)

I valori interi visti come «booleani»

- Risposta:

StudiaBiomedica && (FrequentaPrimoAnno || (! EsamePassato))

Esercizio

Scrivere un programma C che, leggendo da tastiera i valori di verità di tre valori booleani assegnati a ciascuna delle variabili intere `StudiaBiomedica`, `FrequentaPrimoAnno`, `EsamePassato`, **assegni** alla variabile `SegueInformatica` l'espressione di verità dipendente da essi; stampi a video il valore di `SegueInformatica`.

I tipi `float` e `double`

- Entrambi servono per rappresentare i numeri reali, attraverso la **virgola mobile**
- Usare un tipo `float` anziché `double` dipende dal livello di *precisione* necessario al programma
 - Livello del calcolatore → `precisione == numero di bit usati per la rappresentazione`
 - Il `double` usa infatti il doppio dei bit di `float`
 - Ulteriori necessità possono condurre ad usare il tipo `long double`
- Questi tipi sono limitati sia superiormente che inferiormente
 - Limite superiore o di **overflow**: non è possibile rappresentare un valore più grande di...
 - Limite inferiore o di **underflow**: non è possibile rappresentare un valore più piccolo di...
- Trattandosi di valori in virgola mobile, di solito il passaggio da `float` a `double` è motivato da ragioni di underflow

Esercizio

- Scrivere un programma C che stampa a video il quoziente ed il resto della divisione di due numeri **reali** X e Y introdotti da tastiera

- Soluzione

```
/*Programma che stampa a video il risultato della divisione di
due numeri introdotti da tastiera*/
#include <stdio.h>
int main()
{
    float x, y, q;
    scanf("%f %f", &x, &y); /*Notare %f al posto di %d*/
    q = x/y;
    printf("%f / %f = %f", x, y, q);
    return 0;
}
```

Manipolazione di valori numerici

- Come per gli int, sui float e double sono definite le quattro operazioni fondamentali ma l'uscita è differente
 - Esercizio. Scrivere un programma C che stampi a video il risultato della divisione tra due valori interi e gli stessi valori rappresentati come float
- Su tutti i tipi numerici sono definiti operatori di confronto:
 - == uguaglianza
 - (es. $a==b$ restituisce il valore 0 se a e b sono diversi)
 - != diversità (es. $a!=b$)
 - <, > minore, maggiore (es. $a>b$)
 - <=, >= minore o uguale, maggiore o uguale (es. $a<=b$)

Librerie predefinite per la manipolazione di valori numerici

- La Libreria `math.h` mette a disposizione numerose funzioni per la manipolazione di `float` e `double`
- Es.
 - Esponenziali: `pow(x, y) = xy`, `exp(x) = ex`
 - Radice quadrata: `sqrt(x) = x1/2`
 - Logaritmo: `log(x)`, `log10(x)`
 - Trigonometriche: `cos(x)`, `sin(x)`, `tan(x)`, `sinh(x)`, etc...
 - Arrotondamento:
 - `ceil(x)` – minimo intero non minore di `x`: `y = ceil(x) → y ≥ x`
 - `floor(x)` – max intero non maggiore di `x`: `y = floor(x) → y ≤ x`
- Tutte le funzioni restituiscono un valore di tipo `double`

Conversioni double-int e viceversa (il casting)

- Quando le operazioni coinvolgono valori di differente tipo, il C opera una conversione in funzione del tipo a cui la variabile è assegnata
- Per evitare tuttavia errori in fase di calcolo, è meglio effettuare il “casting” ovvero l’operazione di forzatura di conversione di un tipo ad un altro, a mano
- Esempio di casting
 - `int a, b;`
 - `double c;`
 - `c = a + (a/b); /* potrebbe forzare il risultato a/b ad essere convertito come double solo dopo la somma con a*/`
 - `c = a + ((double)a)/((double)b); /*conversione manuale prima della divisione*/`
- In generale l’ordine di conversione dipende dall’implementazione del C, quindi di solito è meglio procedere manualmente
- **Scrivete uno o più programmi C per studiare le differenze in fase di conversione**

Il tipo `char`

- E' il tipo che rappresenta i caratteri secondo il codice ASCII
- Ogni `char` si rappresenta con 8 bit (1 byte), da -127 a 128
- Per assegnare un valore ad una variabile di tipo `char`:
- {
 char carattere;
 carattere='a'; /*il valore è tra due apici*/
 ...
• }

Operazioni su char e varianti

- Sono definite anche le operazioni di confronto (`'a' < 'b'`) e aritmetiche
- `unsigned char` – sempre 8 bit, ma da 0 a 255
- Caratteri speciali
 - `'\n'` : a capo; `'\b'` backspace; `'\t'` tabulazione

Tipo char e codice ASCII

- Visualizzando una variabile `char` come `int`, viene stampato il corrispondente codice ASCII:

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

```
int main()
```

```
{
```

```
    char c;
```

```
    c = 'a';
```

```
    printf("Il codice ASCII del carattere %c è %d\n", c, c);
```

```
    return c;
```

```
}
```

Esercizio

- Scrivere un programma C che legga da tastiera un carattere alfabetico minuscolo e stampi il corrispondente maiuscolo
- Suggerimento. Si osservi la tabella ASCII a lato

Dec	Sym	Dec	Char	Dec	Char	Dec	Char
0	NUL	32		64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	TAB	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	□

Soluzione

```
/*Programma per la lettura e conversione di un carattere*/
#include <stdio.h>
int main()
{
    char c, cM; //c è il carattere minuscolo, cM il corrispondente maiuscolo
    printf("Inserire una lettera dell'alfabeto in minuscolo\n");
    scanf("%c",&c); /*un carattere si legge mediante %c*/
    printf("Il codice ASCII del carattere %c è %d\n",c,c);
    cM = c - ('a' - 'A'); /* si può usare un carattere come intero a 8 bit */
    printf("La corrispondente maiuscola è %c con codice %d\n",cM,cM);
    return 0;
}
```

Nota: la differenza 'a' – 'A' è lo scarto fra la rappresentazione ASCII delle lettere maiuscole e minuscole dell'alfabeto

Definizione di nuovi tipi

- E' possibile definire dei tipi "personalizzati" attraverso la parola-chiave **typedef**
- Esempi
 - **typedef int** anno;
 - **typedef char** carattere;
- A questo punto, possiamo dichiarare delle variabili di tipo `anno` e di tipo `carattere`:
 - `anno x;`
 - `carattere c;`
- In generale la sintassi è:

typedef TipoEsistente NuovoTipo;

La parola-chiave `sizeof`

- Per sapere quanti byte sono necessari per rappresentare un certo tipo, si può usare l'operatore **`sizeof`**
- Esempio. Per sapere quanti byte occupa un `int`, basta scrivere `sizeof(int)`
- Esercizio. Scrivere un programma C che stampi a video il numero di byte necessari per rappresentare i tipi `int`, `short int`, `long int`.
 - L'output dell'operatore **`sizeof`** è di tipo `size_t`, definito nella libreria `<stddef.h>` che è richiamabile mediante la direttiva `#include`.

Soluzione

```
/*Programma che stampa a video la dimensione di int, short int e long int*/
#include <stdio.h>
#include <stddef.h>
int main()
{
    size_t dimint, dimshortint, dimlongint; /*size_t è definito in stddef.h*/
    dimint=sizeof(int);
    dimshortint=sizeof(short int);
    dimlongint=sizeof(long int);
    printf("\nDimensione di int: %d\nDimensione di short int: %d\nDimensione di
    long int: %d\n", dimint, dimshortint, dimlongint);
    return 0;
}
```

I vettori o array

- Con la seguente dichiarazione:

```
int vettore[20];
```

- Dichiariamo una sequenza contigua (in memoria) di 20 variabili dello stesso tipo (in questo caso `int`)
- Si accede al valore della variabile nell'*i*-esima posizione scrivendo `vettore[i]`, con *i* = 0, ..., 19

- Postilla:

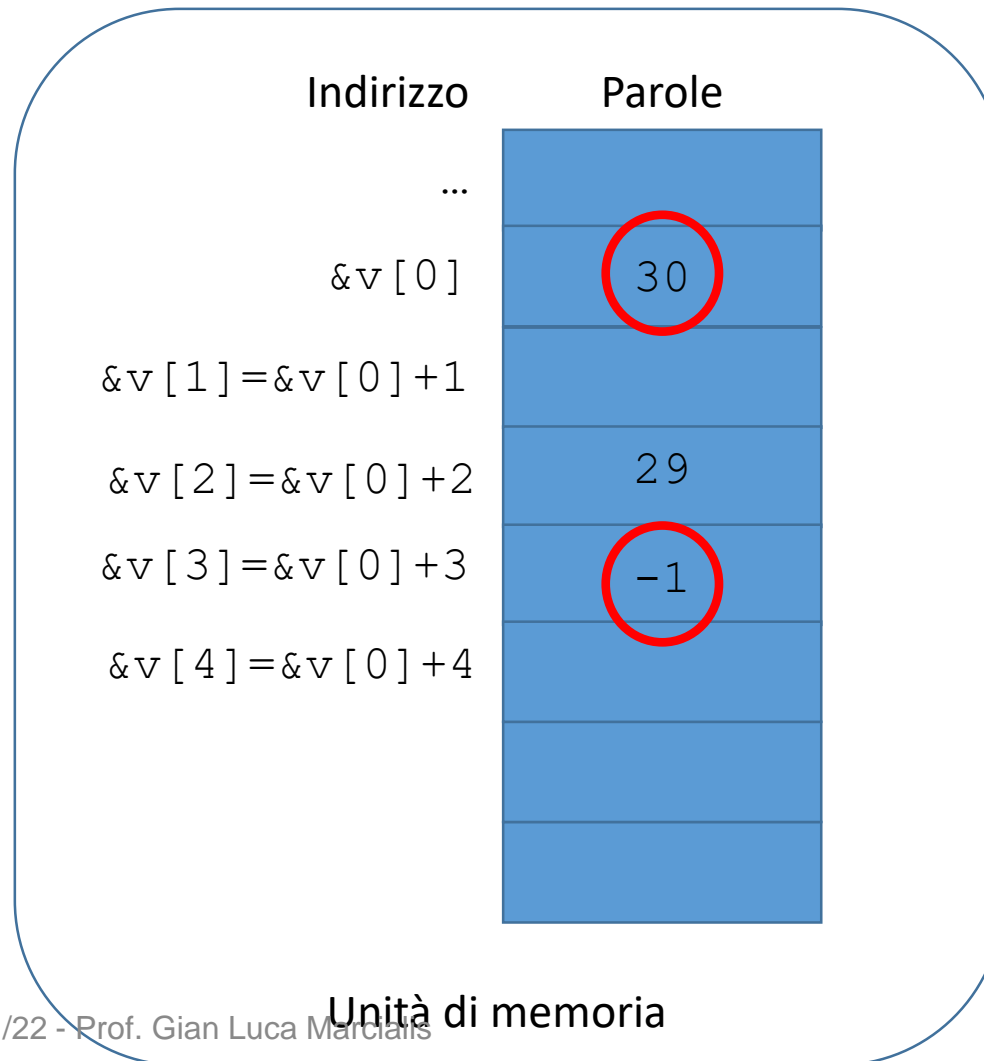
- Possiamo anche ricorrere alla definizione di tipo:

```
typedef int tipo_vettore[20];
```

- Quindi, ora possiamo dichiarare un vettore di 20 interi così:
`tipo_vettore vettore;`

Effetto della dichiarazione di un vettore in memoria ed assegnamento

```
int v[5];
```



```
v[0] = 30;
```

```
v[3] = -1;
```

```
v[2] = v[0] + v[3];
```

Vettori di caratteri: le stringhe

- Quando si definisce un vettore di caratteri con le regole di cui sopra, si parla di **stringhe**
- Es. `typedef char stringa[50];`
 - Definisce un tipo `stringa` consistente in un array di 50 caratteri
- Una stringa, come tale, può rappresentare nomi, frasi, insomma avere una semantica comprensibile all'utente
- Es. dichiariamo due variabili di tipo `stringa` a cui assoceremo il nome ed il cognome di una persona:
 - `stringa nome;`
 - `stringa cognome;`
- L'ultimo carattere di una stringa è sempre assegnato al valore `'\0'`, indipendentemente dalla sua lunghezza massima

Acquisizione e stampa di stringhe

```
char pippo[20];  
char pluto[20]= "CIAO"; /*stringa costante*/  
  
scanf("%s", &pippo[0]);  
/*&pippo[0] è l'indirizzo del primo carattere della  
stringa*/  
  
printf("Stringhe inserite:  %s\t%s\n", pippo, pluto);
```

Esercizio

- Scrivere un programma C che stampa i singoli caratteri della stringa "CIAO" dopo averla assegnata ad una opportuna variabile di tipo stringa.
- In altre parole dovete stampare:

C

I

A

O

Soluzione

```
/*Stampa stringa "CIAO" carattere per carattere */
#include <stdio.h>
int main()
{
    char parola[4]="CIAO";
    /*dichiarazione ed assegnazione contestuale*/

    printf("%c\n",parola[0]);
    printf("%c\n",parola[1]);
    printf("%c\n",parola[2]);
    printf("%c\n",parola[3]);

    return 0;
} /* e se la parola fosse stata precipitevolissimamente? */
```

Indicizzazione della stringa

0	1	2	3	4
<code>'C'</code>	<code>'I'</code>	<code>'A'</code>	<code>'O'</code>	<code>'\0'</code>

Tale indicizzazione vale per le stringhe di qualunque lunghezza.

Operazioni sulle stringhe: libreria **string.h** (`#include <string.h>`)

Operazione * Dato un intero a e le stringhe s, s1, s2, d	Effetto
<code>a=strlen(s);</code>	a contiene la lunghezza di s (il numero di caratteri escluso '\0')
<code>a=strcmp(s1,s2);</code>	a contiene 0 se le due stringhe sono identiche, un numero negativo se s1 precede s2, un numero positivo se s1 segue s2 (ordine ASCII)
<code>strcpy(d,s);</code>	copia la stringa s nella stringa d compreso '\0'
<code>strcat(d,s);</code>	concatena s a d

Esercizio. Scrivere un programma C che, lette due stringhe da tastiera:

- Stampi a video la lunghezza dell'una e dell'altra
- Stampi l'esito del confronto fra le due stringhe
- Copi la seconda stringa nella prima e stampi entrambe le stringhe

Esempi

```
int a;  
char s[100]=" ciao";  
char d[100];  
  
a=strlen(s);  
//a contiene 5 dopo l'esecuzione ➔ " ciao"  
strcpy(d,"buonanotte");  
//d contiene "buonanotte"  
strcat(d,s);  
//d contiene "buonanotte ciao"
```

Per saperne di più

- Ceri, Mandriola, Sbattella, *Informatica – arte e mestiere*, Cap. 5, McGraw-Hill
- Kernighan, Ritchie, *Il linguaggio C*, Cap. 2, Pearson-Prentice Hall