

# Caratteri e Stringhe

# Codice ASCII

'\0' →	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
	00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	`
	00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
	00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
	00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
	00000100	4	End of transmit	00100100	36	\$	01000100	68	D	01100100	100	d
	00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
	00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
	00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
	00001000	8	Backspace	00101000	40	(	01001000	72	H	01101000	104	h
'\t' →	00001001	9	Horizontal tab	00101001	41	)	01001001	73	I	01101001	105	i
'\n' →	00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
'\r' →	00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
	00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
	00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
	00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
	00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
	00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
	00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
	00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
	00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
	00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
	00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
	00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
	00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
	00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
	00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
	00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
	00011011	27	Escape	00111011	59	;	01011011	91	[	01111011	123	{
	00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	
	00011101	29	Group separator	00111101	61	=	01011101	93	]	01111101	125	}
	00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
	00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del

# Caratteri in C

```
int main() {  
    char ch;    // una variabile di tipo char occupa 1 byte  
    ch = 'A';   //dopo questa istruzione ch contiene  
                // il byte 01000001 (65 in decimale)  
    ch = 65;    // questa riga è del tutto  
                // equivalente alla precedente  
    printf( "%c" , ch ); // stampa A  
    printf( "%d" , ch ); // stampa 65  
}
```

# Cos'è una stringa

In generale, in informatica:

Una stringa è una sequenza (finita) di caratteri. Il numero di caratteri è detto lunghezza della stringa.

In particolare, in C:

Una stringa è un array di char terminato dal carattere con codice ASCII 0, detto **NULL CHARACTER** o **TERMINATORE DI STRINGA** e indicato con '`\0`'.

```
int main() {  
    char arr[5] = {'a','b','c'};  
    char str[5] = {'a','b','c','\0'};  
}
```

arr è un array di 5 char,  
ma **NON** è una stringa valida in C !

str è una stringa, perchè termina  
con il carattere '`\0`'

# Lunghezza di una stringa

ATTENZIONE: per **lunghezza** di una stringa si intende è il numero di caratteri che precedono '\0', NON la dimensione dell'array di char.

*Per memorizzare una stringa di lunghezza n occorre un array di almeno n+1 elementi, perchè l'ultimo elemento è occupato dal terminatore di stringa.*

```
int main() {  
    char str[4] = {'a','b','c','\0'};  
}
```



**stringa di lunghezza 3**

# Inizializzazione di una stringa

sintassi speciale per  
inizializzare le stringhe

```
char s[50] = "abc";
```

=

equivalente a

inizializzazione della stringa  
come array, con terminatore

```
char s[50] = {'a','b','c','\0'}
```

E' possibile non specificare la lunghezza ( il compilatore la calcola per noi)

```
char s[] = "abc";
```

=

```
char s[4] = "abc";
```

ATTENZIONE, è un **grave errore** scrivere: `char s[3] = "abc";`

≡


# Perchè usare le stringhe?

Le librerie standard del C contengono funzioni "già pronte" per operare sulle stringhe (cioè su array di char che rispettano la convenzione appena descritta ).

## INPUT/OUTPUT

```
int main() {  
    char parola[31]; // variabile in grado di contenere una parola lunga  
                    // al massimo 30 caratteri ( non 31 !!! )  
    printf(" Inserisci una parola: ");  
  
    scanf(" %s ", parola );  
    printf(" Hai scritto: %s ", parola );  
}
```

**NON mettere & davanti alle stringhe nella scanf poichè il nome della stringa rappresenta già un indirizzo !**



# La funzione gets

**PROBLEMA**: la funzione *printf* interrompe l'acquisizione della stringa appena incontra un carattere "whitespace" ( "spazio" , "a capo" , "tab" ..).

**SOLUZIONE**: si utilizza la funzione **gets**, che interrompe l'acquisizione della stringa quando incontra il carattere "a capo"

```
int main() {  
    char str[MAX];  
    printf(" Inserisci una frase: ");  
    // scanf("%s",str); ← NO: str conterrebbe solo la prima parola  
    gets( str );        ← OK: str contiene l'intera frase inserita  
    printf(" Hai inserito: %s ", str );  
}
```



# La libreria string.h

- strlen      *string length*  
restituisce la lunghezza di una stringa
- strcpy      *string copy*  
copia una stringa in un'altra
- strcat      *string concatenation*  
concatena una stringa a un'altra
- strcmp      *string compare*  
confronta due stringhe per stabilire  
il loro ordine alfabetico

# La libreria string.h

Negli esempi che seguono *s1* ed *s2* sono stringhe: `char s1[MAX],s2[MAX];`

Esempio	Descrizione
<code>int lung;</code> <code>lung = <u>strlen</u>( s );</code>	Restituisce la lunghezza della stringa <i>s</i> .
<code><u>strcpy</u>( s1, s2 );</code>	Ricopia la stringa <i>s2</i> sopra la stringa <i>s1</i> . Il contenuto precedente di <i>s1</i> viene perso.
<code><u>strcat</u>( s1, s2 );</code>	Concatena la stringa <i>s2</i> alla stringa <i>s1</i> , cioè aggiunge alla fine di <i>s1</i> tutti i caratteri di <i>s2</i> .
<code>int ris;</code> <code>ris = <u>strcmp</u>( s1, s2 );</code>	Confronta le due stringhe per stabilire quale viene prima/dopo in ordine alfabetico. Restituisce: <ul style="list-style-type: none"><li>• <i>zero se le due stringhe sono uguali</i></li><li>• <i>un numero positivo se s1 viene dopo s2 in ordine alfabetico.</i></li><li>• <i>un numero negativo se s1 viene prima di s2 in ordine alfabet.</i></li></ul>

# Utilizzo di strlen, strcpy, strcat e strcmp

```
...
int main() {
    char s1[MAX] = "albero", s2[MAX] = "albergo", s3[MAX];
    int n, esito;

    n = strlen(s1);           // n=6

    strcpy( s3, s1 );         // Copia s1 in s3. Ora anche s3 contiene "albero"

    strcat( s3, s2 );         // Concatena s2 a s3. Ora s3 contiene "alberoalbergo"

    esito = strcmp( s1, s2 ); // la variabile esito contiene il risultato del confronto.
    if ( esito == 0 )
        printf(" uguali ");
    else if ( esito > 0 )
        printf(" s1 maggiore di s2 "); // "albero" viene dopo "albergo" in ordine alfabetico
    else
        printf(" s1 minore di s2 ");
}
```

# Prototipo delle funzioni di string.h

```
size_t strlen(const char *str);
```

```
char *strcat(char *dest, const char *src);
```

```
char *strcpy(char *dest, const char *src);
```

```
int strcmp(const char *str1, const char *str2);
```

# Osservazione importante

Tutte le funzioni sulle stringhe ASSUMONO che l'array di char ricevuto come argomento segua la convenzione delle stringhe...

... cioè assumono che l'array contenga, al massimo come ultimo elemento, il carattere terminatore ('\0').

Se ciò non avviene, si hanno gravi errori nel programma ( vedi esempi successivi)

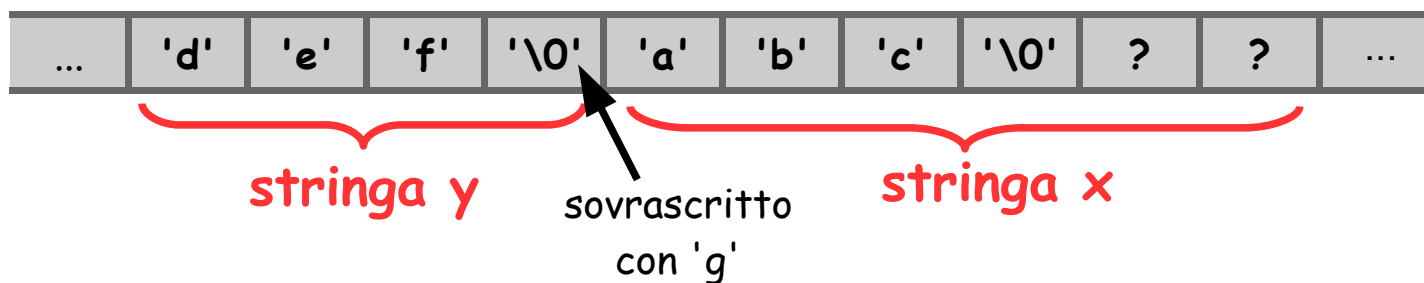
# Osservazione importante (segue)

```
int main() {  
    char x[6] = "abc";  
    char y[4] = "def";  
    y[3] = 'g';  
    printf( "%s", y );  
}
```

il terminatore della stringa y è  
sovrascritto con il carattere 'g' ...

... e questa printf stamperà  
probabilmente "defgabc"

MOTIVO: x e y saranno collocate in memoria RAM come segue<sup>(1)</sup> :



indirizzi di mem.  
crescenti

printf prosegue a stampare finchè non incontra un  
carattere '\0' ( in questo caso quello della stringa x )

(1) La collocazione  
effettiva dipende  
dall'architettura del  
calcolatore e dal  
compilatore

# Stringhe e funzioni