MICROCONTROLLORI – PROGRAMMAZIONE IN C/C++

Nelle applicazioni con i microcontrollori, il passo successivo alla programmazione in assembler, è quello della programmazione con un linguaggio più evoluto che ci consente di avvicinarci ad una modalità d programmazione più lontana dalla macchina più semplice ed intuitiva per il programmatore.

Esistono diversi linguaggi di programmazione, ma nella stragrande maggioranza dei casi, nel mondo dei microcontrollori si prediligono il C ed il C++.

I microcontrollori visti fino ad ora sono quelli della Microchip (cioè i PIC) e quelli della ATMEL (cioè quelli presenti sulla scheda ARDUINO UNO).

In entrambi i casi vengono forniti i relativi ambienti di sviluppo; per la scheda ARDUINO viene fornita la sua I.D.E. mentre per i PIC si possono scegliere svariati ambienti. Nel nostro caso per il microcontrollore PIC è stato scelto l'ambiente di sviluppo fornito dalla ditta serba Mikroelektronika (www.mikroe.com).

Nelle varie spiegazioni ed esempi che seguiranno, cercheremo di utilizzare entrambi gli ambienti di sviluppo.

In entrambi i casi è indispensabile la conoscenza dei rudimenti del linguaggio C e la conoscenza basilare della programmazione ad oggetti del C++.

Questo **non vuol essere un corso di programmazione dei due linguaggi**, ma verranno descritte ed approfondite le parti essenziali per la programmazione del microcontrollore, che possiamo elencare in questo modo:

1. Tipi di variabili e costanti
2. Vettori ed array
3. Puntatori
4. Strutture
5. Operatori
6. Controllo del flusso
7. Funzioni
8. Librerie built in
9. Librerie esterne

Per seguire correttamente questo tutorial, si consiglia di scaricare i due ambienti di sviluppo, o almeno uno dei due in base alle necessità.

IDE ARDUINO https://www.arduino.cc/en/Main/Software

MikroC for PIC https://www.mikroe.com/mikroc-pic

L'ambiente di sviluppo di ARDUINO è open source, e liberamente scaricabile, invece quello della mikroelektronika, è un software proprietario con un ottimo rapporto qualità-prezzo. Rimane comunque possibile scaricare ed utilizzare anche il software della mikroelektronika, in quanto se non acquistato, viene comunque consentito l'uso fino alla generazione di un programma con codice macchina binario inferiore a 2kB.

Ciò significa che per semplici applicazioni, può essere utilizzato liberamente, ma per applicazioni più complesse la compilazione non verrà eseguita.

TIPI DI VARIABILI

Nel linguaggio C e C++, le variabili devono sempre essere dichiarate prima del loro utilizzo.

Dichiarare una variabile, significa indicare al compilatore, il nome della variabile (**identificatore**) ed il suo tipo.

Il compilatore riserverà uno spazio in memoria di dimensioni adeguate a contenere il tipo di dato. Nel linguaggio C i tipi di variabili fondamentali sono i seguenti:

TIPO	SPAZIO OCCUPATO	RANGE DI VALORI	
		min	**max**
• **char**	1 byte	-128	+127
• **unsigned char**	1 byte	0	+255
• **Int** • **short**	2 byte	-32.768	+32.767
• **unsigned** • **unigned int** • **unsigned short**	2 byte	0	+65.535
• **long**	4 byte	-2.147.483.648	+2.147.483.647
• **unsigned long**	4 byte	0	+4.294.967.295
• **float**	4 byte *7 cifre significative* *8 bit esponente* *23 bit mantissa*	$-1,175494351 * 10^{38}$	$3,402823466 * 10^{38}$
• **double**	8 byte *15 cifre significative* *11 bit esponente* *52 bit mantissa*	$-2,2250738585072014 * 10^{308}$	$1,7976931348623158 * 10^{308}$
• **long double**	10 byte *19 cifre significative* *15 bit esponente* *64 bit mantissa*	$-3,36210314311209350863 * 10^{4932}$	$1,18973114953572317650 * 10^{4932}$

I numeri senza virgola, char, int, short e long, vengono memorizzati utilizzando il sistema binario, per comprendere in quale maniera, possiamo considerare il numero più piccolo, cioè il tipo char, con due esempi:

se il numero è positivo: *char prova=98;* *in binario=0110 0010*

se il numero è negativo: *char prova=-98;* *in binario=1001 1110*

Nel primo caso il numero è positivo ed il codice binario è la semplice trasformazione del valore in decimale:

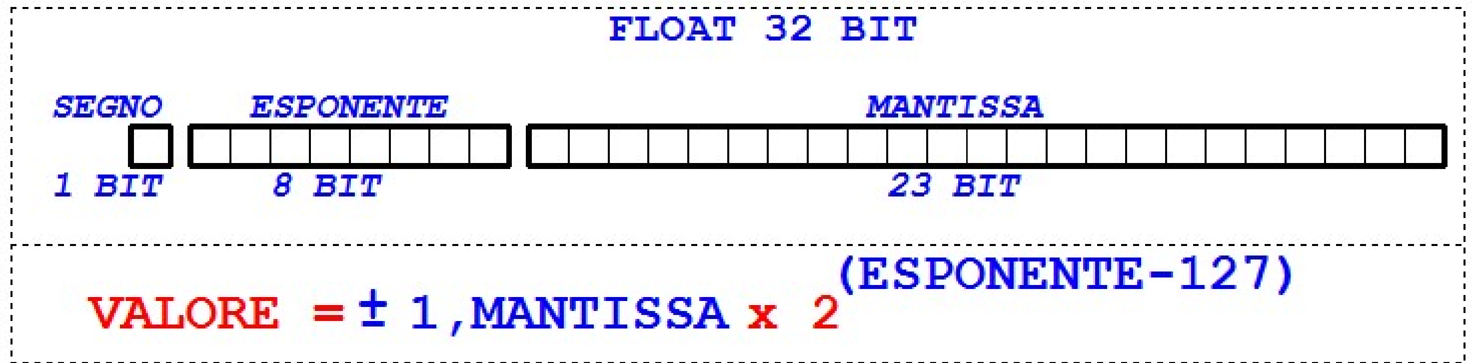
98=0110 0010 = $0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$*=64+32+2=98*

Nel secondo caso il numero negativo viene rappresentato facendo il complemento a due del valore 98 convertito in binario, cioè si ricava il complemento del numero binario invertendo ogni singolo bit e successivamente si somma il valore 1:

complemento di 0110 0010 = 1001 1101

1001 1101 + 1 = 1001 1110

I numeri con la virgola vengono memorizzati utilizzando il ssitema a virgola mobile che prevede una struttura di bit composta da segno-esponente-mantissa. Anche in questo caso consideriamo il più piccolo dei numeri con la virgola e cioè il tipo float:



Lo standard **IEEE754** definisce quanto segue:

SEGNO: *0=positivo 1=negativo*

ESPONENTE: *L'esponente deve rappresentare valori positivi e negativi, per fare questo nello spazio dedicato viene messo il valore dell'esponente reale sommato al valore 127.*
In questo modo volendo memorizzare un esponente pari a +2 ,nel campo troveremo 129 che in binario corrisponde a 1000 0001.
Volento invece memorizzare un esponente pari a -2, nel campo troveremo 125 che in binario corrisponde a 0111 1101.

MANTISSA: *La mantissa è normalizzata, la normalizzaione si ottiene moltiplicando per 2 l'effettivo valore (in binario la moltiplicazione per 2 avviene shiftando a sinistra di una posizione) fino a quando il bit più a sinistra della mantissa diventa 1, eliminando in questo modo tutti i bit a zero non significativi a sinistra. Ciò significa che il bit più a sinistra vale sempre 1, per questo motivo è inutile memorizzare questa informazione, i 23 bit pertanto serviranno solo a rappresentare la parte frazionaria del numero.*
Ad esempio una mantissa pari a ***100 1011 0010 0000 0011 0001*** *trasformata come parte frazionaria in decimale corrisponde a:*

MANTISSA (PARTE FRAZIONARIA)

1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

$$\frac{1}{2^1} + \frac{1}{2^4} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{1}{2^{10}} + \frac{1}{2^{18}} + \frac{1}{2^{19}} + \frac{1}{2^{23}} =$$

$$= 0,58691990375518798828125 + 1 =$$

$$= 1,58691990375518798828125$$

Se ad esempio nei 4 byte abbiamo la seguente sequenza ***0100 0100 0100 1011 0010 0000 0011 0001*** *, significa che il valore sarà ottenuto come segue:*

$$VALORE = \pm 1,M \times 2^{(E-127)}$$

S=0*=+*

E=100 0100 0*=136*

M=100 1011 0010 0000 0011 0001 *=0,586919903755187988828125*

Sostituendo i valori otteniamo:

$$VALORE = +1,586919903755187988828125 \times 2^{(136-127)}$$

VALORE = 812,50299072265625

Al seguente link, un convertitore online dove è possibile verificare e provare la conversione in virgola mobile.

https://www.h-schmidt.net/FloatConverter/IEEE754.html

	Sign	Exponent	Mantissa
Value:	+1	2^9	1.5869199037551884923441
Encoded as:	0	136	
Binary:	☐	☑ ☐ ☐ ☐ ☑ ☐ ☐ ☐	☑ ☐ ☐ ☑ ☐ ☑ ☑ ☐ ☐ ☑ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☑ ☑ ☐ ☐ ☐ ☑

Decimal representation	812.503
Value actually stored in float:	812.50299072265625
Error due to conversion:	
Binary Representation	01000100010010110010000000110001
Hexadecimal Representation	0x444b2031

+1 -1

N.B.

Nel compilatore della mikroelektronica, la conversione in virgola mobile, avviene secondo una tecnica leggermente differente, indicata nella nota Microchip AN575, dove a differenza da quanto previsto dalla IEE754, la posizione del bit del segno è differente e si pone tra l'esponente e la mantissa.

IEEE754 32-bit	sxxx xxxx	y · xxx xxxx	xxxx xxxx	xxxx xxxx
MIcrochip 32-bit	xxxx xxxx	s · xxx xxxx	xxxx xxxx	xxxx xxxx

Il precedente valore convertito con il metodo indicato dalla nota Microchip, avrebbe dato il seguente risultato:

100 0100 0 0 100 1011 0010 0000 0011 0001

Questo è quello che otterremmo assegnando il valore dell'esempio alla variabile pippo, nel compilatore mikroC.

```
float pippo;

void main() {
 pippo=812.50299072265625;
}
```

Edit Value: pippo

1000 1000 0100 1011 0010 0000 0011 0001

Representation

Dec Hex Bin Float Char

La nota Microchip AN575 è disponibile al seguente link: http://ww1.microchip.com/downloads/en/Appnotes/00575.pdf

MIKROC - VARIABILI FONDAMENTALI

Riguardo alle variabili questo ambiente di sviluppo, presente delle piccole differenze, una è quella relativa alla posizione del bit del segno per le variabili in virgola mobile, ma guardando la tabella dei tipi delle variabili estrapolata dall'help del compilatore, si evincono altre differenze rispetto all'ANSI C, che possiamo riassumere nei seguenti punti:

- **Presenza di variabili bit.** Nel linguaggio C non sono previste le variabili che operano sul singolo bit, e volendo operare su un solo bit occorre utilizzare la mascheratura del byte o della word, con operatori logici come AND ed OR. Nell'ambiente mikroC invece è possibile anche operare sui singoli bit.
- **Variabile char senza segno.** Come si può notare una variabile dichiarata char nell'ambiente mikroC non ha segno, a differenza del linguaggio C standard.
- **Variabile short ad 8 bit.** Nel inguaggio C, la variabile short è a 16 bit, nel mikroC è invece ad 8 bit.
- **Variabili float, double e long double della stessa dimensione.** I 3 tipi di variabili, differenti nel linguaggio C, in mikroC hanno la stessa dimensione.
- **Conversione a virgola mobile con metodo Microchip.** Come detto prima la conversione avviene con lo stesso metodo previsto dallo standard IEEE754, con la differenza della posizione del bit di segno.

TIPO	SPAZIO OCCUPATO	RANGE DI VALORI	
		min	max
• **bit**	1 bit	0	1
• **sbit**	1 bit	0	1
• **char** • **unsigned char**	1 byte	0	255
• **signed char**	1 byte	-128	+127
• **short** • **signed short**	1 byte	-128	+127
• **unsigned short**	1 byte	0	255
• **int** • **signed int**	2 byte	-32.767	+32.767
• **unsigned** • **unsigned int**	2 byte	0	+65.535
• **long** • **signed long** • **signed long int**	4 byte	-2.147.483.648	+2.147.483.647
• **unsigned long** • **unsigned long int**	4 byte	0	+4.294.967.295
• **float** • **double** • **long double**	4 byte *8 bit esponente* *23 bit mantissa*	$-1{,}5 * 10^{45}$	$3{,}402823466 * 10^{38}$

ARDUINO - VARIABILI FONDAMENTALI

A differenza del mikroC in questo caso, le differenze rispetto al linguaggio C, sono limitate alla presenza della variabile **boolean**, e della variabile **byte**.

TIPO	SPAZIO OCCUPATO	RANGE DI VALORI	
		min	max
• **boolean**	1 bit	FALSE LOW	TRUE HIGH
• **char**	1 byte	-128	+127
• **unsigned char**	1 byte	0	+255
• **byte** • **usigned byte**	1 byte	0	+255
• **Signed byte**	1 byte	-128	+127
• **Int** • **short**	2 byte	-32.768	+32.767
• **unsigned** • **unsigned int** • **unsigned short**	2 byte	0	+65.535
• **long**	4 byte	-2.147.483.648	+2.147.483.647
• **unsigned long**	4 byte	0	+4.294.967.295
• **float**	4 byte *8 bit esponente* *23 bit mantissa*	$-1{,}175494351 * 10^{38}$	$3{,}402823466 * 10^{38}$
• **double**	8 byte *11 bit esponente* *52 bit mantissa*	$-2{,}2250738585072014 * 10^{308}$	$1{,}7976931348623158 * 10^{308}$

Per assegnare un valore ad una variabile, è sufficiente utilizzare l'**operatore di assegnazione = .**

es. variabile1=10;

Nell'esempio sopra abbiamo assegnato un valore decimale, possiamo comunque assegnare il valore in altri formati:

esadecimale HEX *unsigned char variabile1;* *//dichiaro una variabile unsigned char*

variabile1=0xF4; *//assegna il valore F4 in esadecimale alla variabile1*

binario BIN *unsigned char variabile1;* *//dichiaro una variabile unsigned char*

variabile1=0b11101001; *//assegna il valore 11101010 in binario alla variabile1*

Quando si utilizzano le variabili, bisogna fare molta attenzione a non cadere nell'errore di utilizzare variabili differenti durante le varie operazioni.

Ad esempio se dovessimo dichiarare una variabile di tipo char ed una di tipo int, dobbiamo essere consapevoli che il trasferimento del contenuto da una variabile all'altra può subire una perdia di informazioni, in quanto le dimensioni sono differenti.

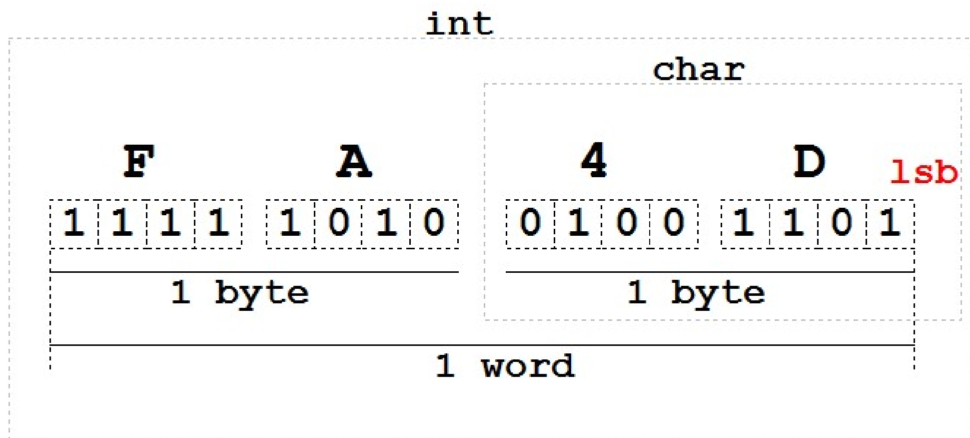
Esempio.

*In questo caso dopo avremo che topo conterrà il valore 0x**4D** e pippo conterrà il valore 0x00**4D***

```
int pippo;
char topo;

void main(){

    pippo=0xFA4D;
    topo=pippo;   //in topo avremo solo
                  //la parte bassa di pippo
}
```

*In questo caso dopo avremo che pippo conterrà il valore 0xFA**4D** e topo conterrà il valore 0x**4D***

```
int pippo;
char topo;

void main(){

    topo=0x4D;
    pippo=topo; //nella parte bassa di pippo
                //avremo tutto il contenuto di topo
}
```



Lo stesso discorso può essere fatto considerando tutte le variabili senza virgola, char, short, int, long.

COSTRUTTO CAST

Capita a volte che nell'effettuare operazioni tra variabili, si hanno risultati indesiderati.

Questo perché vengono utilizzate variabili temporanee di appoggio non adatte per il tipo di risultato.

Nel seguente esempio viene evidenziato un problema di esempio e le possibili soluzioni.

```
void main() {
   int numero1, numero2;
   float risultato;

   numero1 = 1;
   numero2 = 2;
   risultato=numero1/numero2;
   printf("risultato=");
   printf("%f \n",risultato);
}
```

*In questo caso nonostante la variabile **risultato** sia una **float** essa conterrà il **valore 0**, perché viene fatta una divisione tra numeri interi e pertanto v**errà utilizzato temporaneamente un valore int per contenere il risultato.***

SBAGLIATO

```
void main() {
   int numero1, numero2;
   float risultato;

   numero1 = 1;
   numero2 = 2;
   risultato=(float) )numero1/numero2;
   printf("risultato=");
   printf("%f \n",risultato);
}
```

*In questo caso si utilizza i costrutto **cast**, dichiarando prima dell'operazione il tipo di risultato*

float operazione

il risultato sarà 0,5

CORRETTO

```
void main() {
   int numero1, numero2;
   float risultato;

   numero1 = 1;
   numero2 = 2;
   risultato=numero1;
   risultato = risultato/numero2;
   printf("risultato=");
   printf("%f \n",risultato);
}
```

*In questo caso il valore intero **numero1**, viene copiato sulla variabile risultato di tipo **float***
*l'operazione sarà effettuata tra un **float** ed un **int***
ed utilizzerà un numero temporaneo di tipo float
il risultato sarà 0,5

CORRETTO

COSTANTI

Dichiarare una costante significa associare un simbolo ad un valore ed <u>a differenza delle variabili il valore si una costante non cambierà mai.</u>

Le costanti si possono dichiarare in due modi; attraverso la dichiarazione ***const*** o mediante la direttiva ***#define*** .

Una costante <u>non occupa spazio in memoria</u>, in quanto viene gestita solo dal compilatore.

es.

#define PIPPO	*10*	*#define FALSE*	*0*
#define TOPO	*12,5*	*const float*	*pi=3,14*
#define TESTO	*"ciao"*	*const int*	*raggio=100*

All'interno del programma il compilatore sostituirà ogni riferimento ad una costante con il relativo valore indicato.

DICHIARAZIONI E DIRETTIVE

La sezione dichiarativa è quella parte dove vengono dichiarate le variabili e le costanti, la parte dichiarativa può essere **globale** o **locale**, la prima viene posta prima della parte principale del progamma **main()**, e può contenere direttive come ***#include*** e ***#define*** o dichiarazione di variabili e costanti.

La seconda viene invece posta all'interno di una funzione e non può contenere diretive come ***#include*** o ***#define***.

La differenza è sostanziale in quanto ciò che viene dichiarato globalmente vale per tutto il programma comprese le funzioni, mentre invece quello che viene dichiarato localmente vale solo all'interno della funzione.

Una variabile dichiarata globalmente pertanto occuperà spazio sempre durante l'esecuzione del programma, invece una variabile dichirata all'interno di una funzione, occuperà spazio in memoria solo per la durata dell'esecuzione della funzione stessa.

Come detto prima, nella parte dichiarativa globale si possono utilizzare le direttive per il compilatore, vediamo qualche esempio;

#define pi_greco 3,14	*ogni volta che verrà utilizzato il riferimento **pi_greco*** *esso verrà sostituito con il valore 3,14*
#define ever for(; ;)	*Sostituirà al riferimento **ever**, quanto indicato nel **define.*** *Nel programma potremo fare un loop infinito scrivendo:* *ever{* . . . *}*
#define ever while(1)	*Sostituirà al riferimento **ever**, quanto indicato nel **define.*** *Nel programma potremo fare un loop infinito scrivendo:* *ever{* . . . *}*
#include <stdio.h>	*Includerà al programma un file che verrà utilizzato in fase di compilazione.*
#if *#else* *#elif* *#endif*	*Nella parte dichiarativa globale si possono inserire delle condizioni utilizzate dal compilatore mediante queste direttive.* *es.* *#if (espressione1)* *blocco1* *#elif (espressione2)* *blocco2* *#elif (espressione3)* *blocco3* *#endif* *blocco 1 viene eseguito se espressione 1 è vera.* *espressione 2 viene verificata solo se espressione 1 è falsa* *#if (espressione1)* *blocco1* *#else* *blocco2* *#endif* *se espressione 1 è vera esegue blocco1 altrimenti blocco2*

VETTORI ED ARRAY

Un **array** è un insieme di variabili dello stesso tipo, può essere dichiarato allo stesso modo di una variabile indicandone il tipo e la dimensione nel seguente modo:

int pippo[5];

In pratica viene dichiarato un vettore composto da 5 variabili di tipo intero il cui acesso potrà avvenire utilizzando i seguenti identificatori:

tipo[0] tipo[1] tipo[2] tipo[3] tipo[4] tipo[5]

Un array può avere due dimensioni, come una **matrice**, In questo caso la dichiarazione è la seguente:

int pippo[5][4];

Vengono perciò dichiarate 20 variabili su 5 righe e 4 colonne, accessibili con un identificatore dove vanno indicate la riga e la colonna, ad esempio *pippo[3][2]=10;*

E' possibile assegnare dei valori ad un array in fase di sua dichiarazione nei seguente modi:

int pippo[3]={10,20,5}

pippo[0]	10
pippo[1]	20
pippo[2]	5

```
int pippo[3]={10,20,5};
```

⊟ pippo	{...}
[0]	10
[1]	20
[2]	5

int pippo[3][2]={{1,2}, {2,6}, {3,7}};

pippo[0][0]	*1*	*pippo[1][0]*	*2*
pippo[0][1]	*2*	*pippo[1][1]*	*6*
pippo[0][2]	*3*	*pippo[1][2]*	*7*

```
int pippo[3][2]={{1,2},{2,6},{3,7}};
```

Name	Value
⊟ pippo	{...}
⊟ [0]	{...}
[0]	1
[1]	2
⊟ [1]	{...}
[0]	2
[1]	6
⊟ [2]	{...}
[0]	3
[1]	7

Un array di caratteri può essere anche utilzzato per dichiarare una costante di tipo stringa :

```
const char messaggio[]={'h','e','l','l','o'};
const char messaggio[]="hello";
```

Le due dichiarazioni sono identiche, entrambe definiscono la stringa “hello”.

Per assegnare una stringa di caratteri si utilizzano sempre le virgolette, per un singolo carattere gli apici.

CARATTERI E STRINGHE

Un carattere viene rappresentato dal corrispondente numero della tabella ASCII.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
2	SPC 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127
8	€ 128	129	‚ 130	ƒ 131	„ 132	… 133	† 134	‡ 135	ˆ 136	‰ 137	Š 138	‹ 139	Œ 140	141	Ž 142	143
9	144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	˜ 152	™ 153	š 154	› 155	œ 156	157	ž 158	Ÿ 159
A	160	¡ 161	¢ 162	£ 163	¤ 164	¥ 165	¦ 166	§ 167	¨ 168	© 169	ª 170	« 171	¬ 172	- 173	® 174	¯ 175
B	° 176	± 177	² 178	³ 179	´ 180	µ 181	¶ 182	· 183	¸ 184	¹ 185	º 186	» 187	¼ 188	½ 189	¾ 190	¿ 191
C	À 192	Á 193	Â 194	Ã 195	Ä 196	Å 197	Æ 198	Ç 199	È 200	É 201	Ê 202	Ë 203	Ì 204	Í 205	Î 206	Ï 207
D	Ð 208	Ñ 209	Ò 210	Ó 211	Ô 212	Õ 213	Ö 214	× 215	Ø 216	Ù 217	Ú 218	Û 219	Ü 220	Ý 221	Þ 222	ß 223
E	à 224	á 225	â 226	ã 227	ä 228	å 229	æ 230	ç 231	è 232	é 233	ê 234	ë 235	ì 236	í 237	î 238	ï 239
F	ð 240	ñ 241	ò 242	ó 243	ô 244	õ 245	ö 246	÷ 247	ø 248	ù 249	ú 250	û 251	ü 252	ý 253	þ 254	ÿ 255

Una stringa è un array di caratteri e come tale può essere dichiarata indicando la dimensione massima.

char messaggio[10];
messaggio="hello";

dichiarazione di una stringa di dimensione massima 9 caratteri in quanto serve uno spazio per il carattere di terminazione della stringa

La stringa dichiarata sopra conterrà quanto segue

'h'	**'e'**	**'l'**	**'l'**	**'o'**	**0**				

Una stringa può essere dichiarata anche senza indicazione della dimensione:

char messaggio[]="hello";

Anche in questo caso verrà inserito il carattere di terminazione, ma la dimensione utilizzata sarà minore.

'h'	**'e'**	**'l'**	**'l'**	**'o'**	**0**

Nell'ambiente di sviluppo di ARDUINO, esiste anche una classe chiamata String, cosa ben diversa da una funzione che vedremo più avanti.

Termina qui la prima parte dedicata alla programmazione in linguaggio C e C++ negli ambienti mikroC e ARDUINO, nella seconda parte verranno affrontati gli altri argomenti indicati in premessa.