



UNIVERSITÀ  
DI TORINO

---

# Laboratorio di Programmazione I

---

Lezione n. 5:  
Array e stringhe

Alessandro Mazzei

Slides: Elvio Amparore

Un **array** di variabili è una sequenza di tante variabili dello stesso tipo, indicizzate tramite un numero intero non-negativo.

- Il tipo dell'indice è il **size\_t**;
- Gli indici partono da **0**.

In questa prima fase useremo array dichiarati sullo **stack**, che hanno una dimensione prefissata (vedremo più avanti nel corso un altro modo di dichiarare array).

**REMINDER:** lo *specificatore di conversione* usato da printf/scanf per il tipo **size\_t** è **%lu** oppure **%zu**.

Aprire il programma **array.c** , leggere con attenzione il codice, e completare la parte mancante alla fine del main().

# Scrivere codice che usa gli array



Serve sapere qual è la dimensione dell'array.

In questa prima fase avremo:

- Una **#define** che specifica la dimensione, che viene usata ovunque si usa l'array.
- Oppure una variabile che contiene la dimensione dell'array. Servirà quindi avere una coppia:

*<puntatore all'inizio dei dati, numero elementi>*

Tipicamente si opera sugli elementi dell'array tramite cicli.

Ad un campione di 20 soggetti è stato somministrato un questionario, con una domanda che richiede una risposta da 1 a 5 stelle.

**Risposta:** ★★☆☆☆

Le risposte ottenute dai 20 questionari hanno questi valori:

```
#define NUM_RISP 20
int risposte[NUM_RISP] = {
    1, 2, 5, 4, 3, 5, 2, 1, 3, 1,
    4, 3, 3, 3, 2, 3, 3, 2, 2, 5
};
```

Scrivere un programma **questionari.c** che calcola le **frequenze** delle 5 possibili risposte, e ne stampa un sommario. Usare un array **frequenze[]** per accumulare le frequenze delle 5 possibili risposte, prima di stamparle.

Un possibile output è:

Stelle	Frequenza	
1	3	###
2	5	#####
3	7	#####
4	2	##
5	3	###

Sulla pagina Moodle trovate un esercizio con nome



## Lab05-Es1 Somma selettiva

Leggere la specifica e scrivere il codice.

### **DOMANDA:**

- possiamo fare a meno di memorizzare la sequenza letta in un array?

# Array di caratteri: le stringhe



Nel linguaggio C il **testo** è un tipo di dato memorizzato come un **array di elementi di tipo char**.

- Lettere, cifre, simboli sono rappresentati come unità elementari, dette **caratteri**.
- Il tipo **char** è il tipo che rappresenta un carattere.
- Un array di caratteri forma una **stringa** di testo.
- I caratteri corrispondono a codici numerici convenzionali, descritti da degli standard. Lo standard fondamentale è l'**ASCII**<sup>1</sup>.

<sup>1</sup> *American Standard Code for Information Interchange. Si legge 'àski'.*

# I 128 caratteri dell'ASCII



0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39		55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Terminatore:

→ '\0'

Line feed:

→ '\n'

**NUL**

Horizontal tab: **HT** → '\t'

Carriage return: **CR** → '\r'

**LF**



## Un po di storia della codifica del testo nei sistemi digitali.

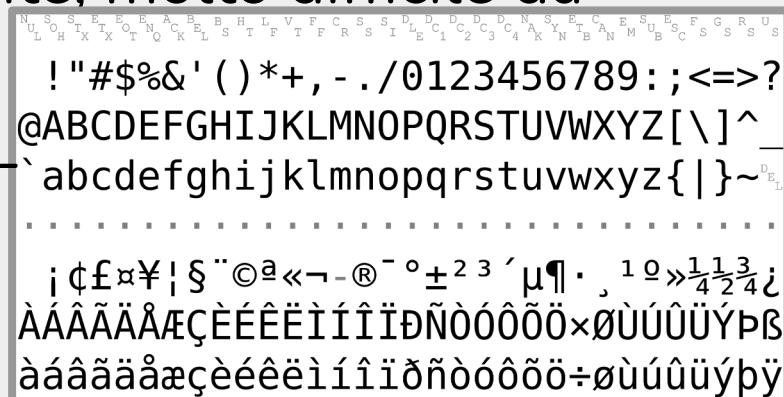
### 1963: ASCII.

- Codifica a 7bit, permette di codificare a malapena gli alfabeti Latini.
- È subito evidente che non è sufficiente per codificare la complessità degli alfabeti in uso nel mondo (*greco, cirillico, arabo, bengalese, tamil, devanagari, ...*) e ancora meno adatto per i tre alfabeti ideografici (**CJK** - *cinese, giapponese, coreano*). Mancano inoltre molti simboli utili

Φ Ϻ غ ૨ ஹ ཨ 碼 人 법 Ε ∇ ∈ ∞

## Anni 70-80: Le Codepages.

- Per non consumare troppo spazio, emerge l'idea di usare i 128 caratteri aggiuntivi a seconda di un flag di sistema.
- Il sistema può quindi usare un set di caratteri esteso a 8bit, detto *codepage*, ma non più di uno alla volta.
- Un file di testo può essere interpretato correttamente solo sapendo per quale codepage è stato pensato.
- Più di **1400** codepages definite, molto difficile da gestire.
- La più usata in EU/US è la **ISO/IEC 8859-1** (detta *Latin-1*), che aggiunge le lettere accentate.



! " # \$ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?  
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ \_  
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~  
.....  
ı ċ ₣ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿  
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß  
à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

## 1988: Unicode 1.0

- Prima proposta di sistema che codifica caratteri a 16bit e non a 8, e che non usa codepages ma standardizza e unifica tutti i codici dei caratteri (detti *codepoints*).
- 16bit sembrano tanti ( $65 \cdot 536$  valori), ma in verità sono insufficienti - ad esempio solo gli **ideogrammi unificati CJK** sono  $97 \cdot 680$ .
- I primi 256 caratteri sono ASCII + Latin-1
- Usa 16bit per carattere, che è meno efficiente.



## 1996: Unicode 2.0 e successivi.

- Definisce 149'813 codepoints a 21bit (max 2'097'152), organizzati in 161 alfabeti. Include tutto il **CJK**.
- Purtroppo è meno efficiente in spazio.

Per ovviare al problema, Unicode separa la **definizione** dalla **codifica** (encoding).

- Codifiche **wide-character**:
  - **UCS-16**: 16bit per carattere (subset)
  - **UCS-32**: 32bit per carattere (completo ma occupa molto spazio)
- Codifiche **multi-byte**:
  - **UTF-8**: dati su 8bit, ogni carattere occupa uno spazio variabile da 1 a 6 bytes. Efficiente ma complesso da gestire.



# Testo in programmazione 1



Il linguaggio C supporta 3 codifiche per il testo:

- **single-byte:** **char**, va bene per ASCII e Latin-1
- **wide:** **wchar\_t**
- **multi-byte:** sempre **char** ma con interpretazione più elaborata per identificare i caratteri

Nel corso di Programmazione 1 useremo solo il testo **single-byte**, per semplicità, e non considereremo le altre codifiche.

# Terminatore della stringa

Una stringa ha una differenza fondamentale rispetto agli array generici: per convenzione è **terminata dal carattere ' \0 ', detto terminatore o NUL.**

Ad esempio la stringa di testo “ciao” è un array di 5 elementi di tipo **char** con questi valori.

'c'	'i'	'a'	'o'	' \0 '
-----	-----	-----	-----	--------

Una stringa di testo sarà quindi dichiarata come:

```
char txt[] = { 'c', 'i', 'a', 'o', ' \0 ' };
```

oppure come puntatore

```
char *txt = "ciao"; // il compilatore ha aggiunto il terminatore
```

Aprire il programma **stringhe.c** , leggere con attenzione il codice, e provare ad eseguirlo.

Scrivere un programma **alfabeto.c** che crea un array di 27 caratteri.

- Il programma inizializza con un ciclo for i primi 26 caratteri con le lettere progressive dell'alfabeto dalla 'A' fino alla 'Z'.
- Inserisce il terminatore della stringa.
- Stampa la stringa ottenuta con printf.



Sulla pagina Moodle trovate un esercizio con nome



**Lab05-Es2 Montagne**

Leggere la specifica e scrivere il codice.  
Leggere con attenzione i suggerimenti.



# Quantificazione universale



Supponiamo che stiamo scrivendo un programma che opera su di un **array** `a[i]` di lunghezza **len**, e vogliamo rispondere a questa domanda:

*Tutti gli elementi dell'array soddisfano una condizione **Cond**?*

Come possiamo fare?

```
bool tutti = true;
for (size_t i=0; i<len && tutti; i++) {
    if (! Cond valutata su a[i] )
        tutti = false;
}
```

**Variabile sentinella**  
(sentry variable)

Il ciclo termina sotto due condizioni

- **i==len**  $\Rightarrow$  tutti gli elementi soddisfano *Cond*
- **tutti==false**  $\Rightarrow$  non tutti gli elementi soddisfano *Cond*.  
In questo caso l'elemento `a[i]` è il primo che non soddisfa.

Nessun elemento dell'array soddisfa una condizione *Cond*?

```
bool nessuno = true;
for (size_t i=0; i<len && nessuno; i++) {
    if (Cond valutata su a[i])
        nessuno = false;
}
```

Il ciclo termina sotto due condizioni

- **i==len**  $\Rightarrow$  nessun elemento soddisfa *Cond*
- **nessuno==false**  $\Rightarrow$  non tutti gli elementi non soddisfano *Cond*.  
In questo caso l'elemento **a[i]** è il primo che soddisfa.

# Quantificazione esistenziale



Supponiamo che stiamo scrivendo un programma che opera su di un **array** `a[i]` di lunghezza **len**, e vogliamo rispondere a questa domanda:

Esiste almeno un elemento dell'array  
che soddisfa una condizione *Cond*?

```
bool esiste = false;
for (size_t i=0; i<len && !esiste; i++) {
    if (Cond valutata su a[i])
        esiste = true;
}
```

Il ciclo termina sotto due condizioni

- `i==len`  $\Rightarrow$  non esiste un elemento che soddisfa *Cond*
- `esiste==true`  $\Rightarrow$  esiste almeno un elemento che soddisfa *Cond*.

Esiste un elemento dell'array  
che non soddisfa una condizione *Cond*?

```
bool controesempio = false;  
for (size_t i=0; i<len && !controesempio; i++) {  
    if (! Cond valutata su a[i] )  
        controesempio = true;  
}
```

Il ciclo termina sotto due condizioni

- **i==len**  $\Rightarrow$  non esiste un elemento che non soddisfa *Cond*
- **controesempio==true**  $\Rightarrow$  esiste almeno un elemento che non soddisfa *Cond*, e l'elemento *a[i]* è il primo che non soddisfa.

# Funzioni base per i caratteri



La libreria del linguaggio C dispone di diverse funzioni per classificare la natura dei caratteri, e manipolarli.

Sono definite includendo l'header **<ctype.h>**

Nome	Descrizione
<b>bool</b> <code>isalpha(char ch)</code>	Controlla se il carattere è alfabetico.
<b>bool</b> <code>isdigit(char ch)</code>	Controlla se il carattere è una cifra.
<b>bool</b> <code>isalnum(char ch)</code>	Controlla se il carattere è alfanumerico.
<b>bool</b> <code>islower(char ch)</code>	Controlla se il carattere è una lettera minuscola.
<b>bool</b> <code>isupper(char ch)</code>	Controlla se il carattere è una lettera maiuscola.
<b>bool</b> <code>isspace(char ch)</code>	Controlla se il carattere è uno spazio, una tabulazione o un ritorno a capo.
<b>bool</b> <code>ispunct(char ch)</code>	Controlla se è un carattere di punteggiatura tra i seguenti: ! " # \$ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` {   } ~
<b>char</b> <code>tolower(char ch)</code>	Se il carattere è alfabetico, ritorna la forma minuscola, altrimenti ritorna il carattere in input inalterato.
<b>char</b> <code>toupper(char ch)</code>	Come tolower ma ritorna la forma maiuscola.

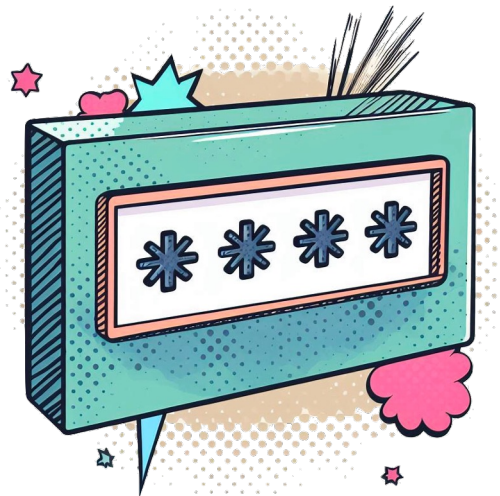
Sulla pagina Moodle trovate un esercizio con nome



## Lab05-Es3 Passwords

Leggere la specifica e scrivere il codice.

Leggere con attenzione quali sono le cinque proprietà da calcolare per ciascuna password in input.



# Indovina Chi?

*Indovina Chi?* è un famoso gioco da tavolo inventato da Theo e Ora Coster nel 1980 in Gran Bretagna dall'azienda MB con il nome *Guess Who?*.

Sulla pagina Moodle trovate un esercizio con nome



## Lab05-Es4 Indovina chi?

completate l'esercizio descritto.

