PROGRAMMAZIONE 2: SPERIMENTAZIONI

Lezione 5 – Caratteri e stringhe in C

Agenda

- Introduzione
- I caratteri
  - ASCII
  - Input/Output di caratteri
  - Manipolazione dei caratteri (<ctype.h>)
- Le stringhe
  - Stringhe immutabili
  - Input/Output di stringhe
  - Manipolazione di stringhe (<string.h>)

Video lezioni disponibili su YouTube

Introduzione

- In questa lezione saranno introdotte le funzioni della libreria standard del C che facilitano l'elaborazione di stringhe e caratteri.
- Le funzioni prese in esame permettono di elaborare caratteri, stringhe e righe di testo.
- Anche gli editor, i word processor e altri tipi di software per l'elaborazione dei testi si basano sulle nozioni fondamentali legate a stringhe e caratteri.

I caratteri (char)

- Un programma può contenere **costanti carattere**.
- Una **costante carattere** è un **valore intero** (`int`) rappresentato come un carattere tra **apici singoli** `' '`.
- Il valore intero del carattere è definito dall'insieme dei caratteri della macchina su cui gira il programma.
- Ad esempio, nello standard **ASCII**, `'z'` corrisponde all'intero 122 come lo `'\n'` corrisponde al numero 10.

I caratteri (char)

- Sintatticamente, i `char` non sono altro che degli `int` di piccola dimensione (un byte, ovvero otto bit).
- Ogni operazione possibile su un `int`, è anche possibile su un `char`.
  - Ovviamente solo alcune di tali operazioni avranno senso sull'interpretazione testuale (**ASCII**) del valore numerico.

```
char a = 'A';    // ASCII: 65
char b = 65;     // ASCII: 'A'
a = a + 1;       // a diventa 66 quindi 'B'
b = b * 2        // non ha molto senso
if (a == 65)     // falso
if (b == 'B')    // vero
```

ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

http://www.asciitable.com

Input/Output di caratteri

- È possibile ricevere in **input** un carattere tramite la funzione `getchar()` (definita in `<stdio.h>`) che legge un dato da tastiera e lo restituisce come `int`.

```
char a, temp;
printf("Inserire un carattere: ");
a = getchar();
temp = getchar(); // consuma l'invio
```

- Si noti l'uso di `temp` perché l'utente deve **sempre** premere **invio** (**return**), anche se il programma richiede un singolo carattere;
  - ricordare che, se l'utente inserisce più caratteri, questi verranno restituiti uno ad uno nelle `getchar` successive, quindi la seconda `getchar` conterrà l'invio.

Input/Output di caratteri

- È possibile restituire in **output** un carattere tramite la funzione `putchar()` (definita in `<stdio.h>`) che legge il valore di una varabile di tipo char e lo restituisce come `int`.

```
char a, temp;
printf("Inserire un carattere: ");
a = getchar();
temp = getchar(); // consuma l'invio
printf("Caratteri inseriti: \n");

// visualizza a video il carattere contenuto in a
putchar(a);

int x = putchar(a);
// visualizza il carattere contenuto in a e salva in x il
suo valore in int (in base all'ASCII)
```

Input/Output di caratteri

es1.c

```c
int main(void)
{
  char a, temp;
  printf("Inserire un carattere: ");
  a = getchar();
  temp = getchar(); // consuma l'invio
  printf("Carattere letto (char): %c\n",a);
  printf("Carattere letto (int): %d\n",a);
  printf("Carattere letto (char): %c",temp);
  printf("Carattere letto (int): %d\n",temp);
  printf("Carattere letto (char): ");
  // putchar visualizza a video il carattere in a
  // e salva il suo valore ASCII in x
  int x = putchar(a);
  printf("\n");
  printf("Carattere letto (int): %d\n",x);
}
```

Manipolazione dei caratteri (<ctype.h>)

- La libreria di funzioni per il trattamento dei caratteri è definita in `<ctype.h>`.
- La libreria include diverse funzioni che eseguono utili test e manipolazioni di dati di tipo carattere.
- Ogni funzione riceve come argomento un `unsigned char` (ovvero un `int`) o `EOF` (costante simbolica che vale -1).
- In un programma che riceve dati dagli utenti, può tornare molto utile inserire controlli e manipolazioni per migliorarne la funzionalità.

Libreria <ctype.h>

Nome	Descrizione
Funzioni per la verifica delle proprietà dei caratteri: ritornano zero se falso o un numero diverso da zero se vero	
`isalnum`	Controlla che il carattere passato sia *alfanumerico*.
`isalpha`	Controlla che il carattere passato sia *alfabetico*.
`isblank` (introdotto dal C99)	Controlla che il carattere passato sia *bianco*, cioè non visibile a schermo (spazio o tabulazione).
`iscntrl`	Controlla che il carattere passato sia *di controllo*.
`isdigit`	Controlla che il carattere passato sia *numerico* (non dipendente dalle impostazioni locali).
`isgraph`	Controlla che il carattere passato sia *grafico*, cioè abbia un glifo ad esso associato. I caratteri di spaziatura, ad esempio, non sono considerati *grafici*.
`islower`	Controlla che il carattere passato sia *minuscolo*.

Libreria <ctype.h>

Nome	Descrizione
Funzioni per la verifica delle proprietà dei caratteri: ritornano zero se falso o un numero diverso da zero se vero	
`isprint`	Controlla che il carattere passato sia *stampabile*.
`ispunct`	Controlla che il carattere passato sia *di punteggiatura* (es.: $, #, ?, ecc...).
`isspace`	Controlla che il carattere passato sia *di spaziatura*.
`isupper`	Controlla che il carattere passato sia *maiuscolo*.
`isxdigit`	Controlla che il carattere passato sia *esadecimale*, cioè sia compreso in 0-9, oppure a-f, oppure A-F.
Funzioni per la conversione dei caratteri: ritornano il carattere convertito	
`tolower`	Converte il carattere passato nel suo corrispondente minuscolo, se applicabile.
`toupper`	Converte il carattere passato nel suo corrispondente maiuscolo, se applicabile.

Libreria <ctype.h>

es2.c

```c
#include <ctype.h> // funzioni sui caratteri

int main(void)
{
  // carattere inserito dall'utente
  char c;

  int c_low = 0, c_upp = 0, c_dig = 0, c_alp = 0, c_spa = 0, c_pun = 0;

  printf("Inserire una password: \n");

  // finché non è stato dato l'invio (return)
  while( (c=getchar()) != '\n' )
  {
    if (isdigit(c)) // cifre
      c_dig++;
    if (isalpha(c)) // lettere
      c_alp++;
    if (islower(c)) // minuscole
      c_low++;
    if (isupper(c)) // maiuscole
      c_upp++;
    if (isspace(c)) // spazi
      c_spa++;
    if (ispunct(c)) // simboli stampabili diversi dallo spazio
      c_pun++;
  }

  printf("\n");
  printf("Cifre: %d \n",c_dig);
  printf("Lettere: %d \n",c_alp);
  printf("Maiuscole: %d \n",c_upp);
  printf("Minuscole: %d \n",c_low);
  printf("Spazi: %d \n",c_spa);
  printf("Caratteri speciali: %d \n",c_pun);

  return 1;
}
```

Libreria <ctype.h>

es2.c

```
Inserire una password:
Ciao Mamma #! 88

La password "Ciao Mamma #! 88" contiene:
Cifre: 2
Lettere: 9
Maiuscole: 2
Minuscole: 7
Spazi: 3
Caratteri speciali: 2
```

Le stringhe

- Una **stringa** è una serie di caratteri trattati come singola unità.
- Una stringa **può** contenere lettere, cifre e vari caratteri speciali come `+`, `-` , `*` ecc...
- Le stringhe sono scritte tra **apici doppi** `" "`.
- Ogni stringa in C è terminata dal **carattere nullo** `'\0'`.
- A una stringa si accede tramite il puntatore al primo carattere della stringa stessa.
  - Pertanto è corretto dire che una **stringa è un puntatore** (al primo carattere).
  - Pertanto, le **stringhe sono array (vettori) di caratteri**.

Le stringhe

- È possibile inizializzare una stringa in un **array di caratteri** o una **variabile di tipo `char *`**.

```
char colore[]= "blu";
char *colorePtr = "blu";
```

- La prima definizione dichiara un array di caratteri di quattro elementi (`'b','l','u','\0'`) e determina automaticamente la dimensione dell'array.
- Le seconda definizione dichiara un puntatore a una stringa `"blu\0"` memorizzata da qualche parte in memoria (il puntatore punterà al primo carattere `'b'`).
  - Le stringhe definite tramite puntatore sono dette **letterali immutabili** (in quanto non modificabili).

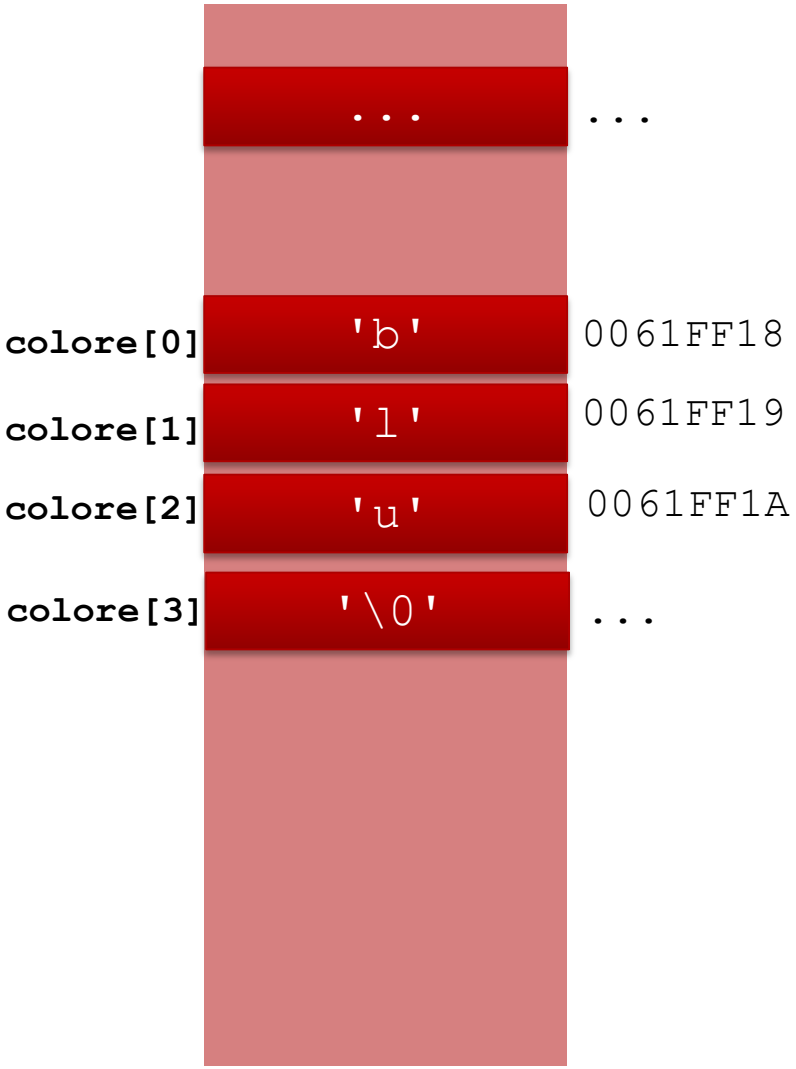
Le stringhe

es3.c

```c
int main(void)
{
  char colore[]= "blu";
  char *colorePtr = "blu";
  int i;

  // di fatto l'array è un puntatore
  printf("Indirizzo array: %p\n",colore); // è possibile omettere la &
  i = 0;
  // scorre la prima stringa e determina gli indirizzi in memoria
  while(colore[i]!='\0')
  {
    printf("Carattere %c (ASCII %d) -> indirizzo di memoria %p\n", colore[i], colore[i], &colore[i]);
    i++;
  }

  printf("\n");
  // scorre la seconda stringa
  // &colorePtr restituisce l'indirizzo di memoria in cui è salvato il puntatore
  // colorePtr contiene l'indirizzo di memoria del primo carattere della stringa
  // quindi è un puntatore a un carattere
  // *colorePtr accede al contenuto puntato
  printf("Indirizzo del puntatore alla stringa: %p\n",&colorePtr);
  printf("Indirizzo puntato inizialmente dal puntatore: %p\n",colorePtr);
  while(*colorePtr!='\0')
  {
    printf("Carattere %c (ASCII %d) -> indirizzo di memoria %p \n", *colorePtr, *colorePtr, colorePtr);
    //(*colorePtr)++; // ERRORE! Cerca di modificare il contenuto
    // sposta il puntatore all'elemento successivo
    colorePtr++;
  }
  return 1;
}
```

Le stringhe

```
Indirizzo array: 0061FF18
Carattere b (ASCII 98) -> indirizzo di memoria 0061FF18
Carattere l (ASCII 108) -> indirizzo di memoria 0061FF19
Carattere u (ASCII 117) -> indirizzo di memoria 0061FF1A

Indirizzo del puntatore alla stringa: 0061FF14
Indirizzo puntato inizialmente dal puntatore: 00405044
Carattere b (ASCII 98) -> indirizzo di memoria 00405044
Carattere l (ASCII 108) -> indirizzo di memoria 00405045
Carattere u (ASCII 117) -> indirizzo di memoria 00405046
```



Le stringhe

```
Indirizzo array: 0061FF18
Carattere b (ASCII 98) -> indirizzo di memoria 0061FF18
Carattere l (ASCII 108) -> indirizzo di memoria 0061FF19
Carattere u (ASCII 117) -> indirizzo di memoria 0061FF1A

Indirizzo del puntatore alla stringa: 0061FF14
Indirizzo puntato inizialmente dal puntatore: 00405044
Carattere b (ASCII 98) -> indirizzo di memoria 00405044
Carattere l (ASCII 108) -> indirizzo di memoria 00405045
Carattere u (ASCII 117) -> indirizzo di memoria 00405046
```

Le stringhe

Il C standard indica che una stringa dichiarata come puntatore è immutabile (cioè non modificabile), ma questa regola non vale per tutti i compilatori**. Se è necessario modificare una stringa, dichiararla come array di caratteri**.

Il C permette la memorizzazione di stringhe di qualsiasi lunghezza. Si consiglia quindi di dichiarare array abbastanza grandi da contenere la stringa più lunga memorizzabile.

Ricordarsi sempre di allocare uno spazio per il carattere nullo di fine stringa (`'\0'`).

Stringhe immutabili

- Le stringhe definite tramite puntatore sono dette **letterali immutabili** (in quanto non modificabili).
- Cercare di modificare una stringa comporta un errore in fase di esecuzione

```
while(*colorePtr!='\0')
{
   printf("Carattere %c indirizzo %p \n", *colorePtr, colorePtr);
//  ERRORE! accede al contenuto e cerca di aumentarlo di 1
// (ovvero sposta in avanti di 1 la sua codifica ASCII)
    (*colorePtr)++;
    // sposta il puntatore all'elemento successivo
    colorePtr++;
}
```

```
Seconda stringa: 0x7ffeea9b7b30
Carattere b indirizzo 0x105248f5e
Bus error: 10
```

Input/Output di stringhe

- Sono presenti diverse funzioni per l'input/output di stringhe nella libreria `<stdio.h>` (in **grassetto** le variabili che conterranno il risultato delle funzioni).

Nome	Descrizione
La presenza del prefisso `const` (costante) indica che la stringa passata come parametro non è modificabile (quindi può essere una stringa costante `"..."`, un puntatore o un array).	
`char *fgets(char *s, int n, FILE *stream)`	Legge una sequenza di caratteri **dallo** `stream` specificato **e li memorizza nella stringa** `s` finché non viene incontrato un newline (o EOF) oppure finché non vengono letti `n-1` byte (1 byte = 1 carattere). Il carattere di terminazione `'\0'` è aggiunto in automatico a fine stringa.
`int puts(const char *s)`	Visualizza una stringa `s` seguita dal newline. Restituisce 1 se non ha incontrato errori (0 altrimenti).
`int sprintf(char *s, const char *format, ...dati...)`	Come la `printf` ma anziché visualizzare la stringa a video, la salva nella stringa `s` in base ai `dati` ricevuti.
`int sscanf(char *s, const char *format, ...variabili...)`	Come la `scanf` ma i valori anziché essere letti da tastiera, vengono letti **da** `s` e salvati nelle variabili.

scanf

- È possibile memorizzare una stringa in un array usando la `scanf`.
- La `scanf` legge caratteri finché non incontra uno spazio, una tabulazione, una newline (invio) o la fine di un file.

```
char nome[19]; // 0 .. 18
printf("Inserisci il tuo nome: ");
scanf("%18s", nome);
```

- La stringa inserita dall'utente sarà salvata nella variabile `nome`.
- Siccome `nome` è un array (quindi un puntatore), la `&` **non** è necessaria.
- Si noti la notazione `%18s` a indicare il numero massimo di caratteri accettati; il valore è inferiore di 1 rispetto alla lunghezza massima per consentire l'inserimento dello `'\0'`.

scanf

es4.c

```
6  int main(void)
7  {
8    char nome[19]; // 0 .. 18
9    int i;
10
11   printf("Inserisci il tuo nome: ");
12   scanf("%18s", nome);
13   printf("Stringa inserita: %s\n", nome);
14
15   // scorre la stringa
16   i = 0;
17   while(nome[i]!='\0')
18   {
19     printf("Carattere %d: %c \n", i, nome[i]);
20     i++;
21   }
22   return 1;
23 }
```

scanf

```
Inserisci il tuo nome: Andrea Maria Luigi
Stringa inserita: Andrea
Carattere 0: A
Carattere 1: n
Carattere 2: d
Carattere 3: r
Carattere 4: e
Carattere 5: a
```

La stringa è stata letta sino allo spazio, quindi tutti i caratteri dopo "Andrea" sono stati ignorati

gets e puts

- La funzione `gets` legge una serie di caratteri dallo standard input (**stdin** ovvero la rappresentazione in file dei dati ricevuti da tastiera) e li memorizza nel primo argomento (la stringa) finché non incontra un invio (newline) o EOF (end-of-file) oppure finché non viene letto il numero massimo - 1 dei caratteri consentiti. Il carattere nullo di fine stringa '\0' viene inserito automaticamente in fondo alla stringa quando termina la lettura.
- La funzione **puts** visualizza una stringa a video seguita dall'invio (newline).

gets e puts

es5.c

```c
int main(void)
{
  char nome[SIZE]; // 0 .. 31

  puts("Inserisci il tuo nome:");

  // legge al massimo SIZE-1 caratteri da stdin (la tastiera) e li salva in nome
  fgets(nome, SIZE, stdin);

  puts("Stringa ricevuta:");
  puts(nome); // simile a printf("%s", nome);

  return 1;
}
```

sprintf

- La funzione `sprintf` memorizza in una stringa s una serie di dati formattati.
- La funzione **usa lo stesso sistema di formattazione** della `printf` (`%c`, `%d`, ecc...).
- La creazione di stringhe contenenti dati aggregati è spesso utilizzata per lo scambio di dati tra applicazioni diverse.

sprintf

es6.c

```
7   int main(void)
8   {
9     char *nome = "Andrea";
10    int eta = 20;
11    float peso = 75.88;
12
13    char frase[SIZE];
14
15    // legge i dati (dal terzo parametro in poi) e li salva formattati in stringa
16    sprintf(frase, "%s;%d;%.2f", nome, eta, peso);
17
18    puts("Stringa creata:");
19    puts(frase);
20
21    return 1;
22  }
```

sscanf

- La funzione `sscanf` legge una stringa contenente una serie di dati formattati e memorizza il risultato in una serie di variabili.
- La funzione **usa lo stesso sistema di formattazione** della `scanf` (`%c`, `%d`, ecc...).
- La funzione separa i dati contenuti nella stringa in base agli spazi (es.: `"7 8 9"` sono tre dati mentre `"789"` è un dato solo).

sscanf

es7.c

```c
#define SIZE 64

int main(void)
{
  char *frase = "Andrea 20 75.88";
  char nome[SIZE]; // non può essere un char *
  int eta;
  float peso;

  // legge la frase (primo parametro) e salva i dati formattandoli nel rispettivo formato
  // lo spazio tra i formati è inutile rende solo più leggibili i formati
  // si noti la & per le variabili eta e peso (come per scanf)
  // si noti che nome è stato dichiarato array per essere modificabile
  sscanf(frase, "%s %d %f", nome, &eta, &peso);

  printf("Dati ricavati dalla stringa:\n");
  printf("nome: %s\n", nome);
  printf("eta: %d\n", eta);
  printf("peso: %f\n", peso);

  return 1;
}
```

Manipolazione di stringhe (<string.h>)

- La libreria per la manipolazione di stringhe è `<string.h>`.
- La libreria fornisce molte funzioni utili per **copiare** stringhe, **concatenarle**, **confrontarle**, **suddividerle** in token (parti logiche) e determinarne la **lunghezza**.
- **Ogni funzione descritta aggiunge il carattere nullo '\0' di fine stringa in coda al risultato**.
- In questa lezione si vedranno alcune delle funzioni più importanti.

Manipolazione di stringhe (<string.h>)

Nome	Descrizione
La presenza del prefisso `const` (costante) indica che la stringa passata come parametro non è modificabile (quindi può essere una stringa costante `"..."`, un puntatore o un array).	
`char *strcpy(char *s1, const char *s2)`	Copia la stringa `s2` in `s1`. Viene restituito il valore di `s1`.
`char *strcat(char *s1, const char *s2)`	Aggiunge la stringa `s2` in `s1`. Il primo carattere di `s2` sovrascrive il carattere di fine stringa `'\0'` di `s1`. Viene restituito il valore di `s1`.
`int strcmp(const char *s1, const char *s2)`	Confronta `s1` con `s2`. La funzione **restituisce 0 se le due stringhe sono uguali** oppure un numero positivo o negativo se `s1` è maggiore o minore di `s2`.
`char *strchr(const char *s, int c)`	Cerca la **prima occorrenza** del carattere `c` nella stringa `s`. Se `c` viene trovato, viene restituito un puntatore a quel carattere delle stringa, altrimenti `NULL`.
`char *strrchr(const char *s, int c)`	Come `strchr` ma cerca **l'ultima occorrenza** di `c`.
`char *strstr(const char *s1, const char *s2)`	Cerca la prima occorrenza della stringa `s2` in `s1`. Se la stringa viene trovata, viene restituito un puntatore all'inizio di quella stringa in `s1`, `NULL` altrimenti.
`char *strtok(char *s1, const char *s2)`	Una sequenza di chiamate a strtok suddivide la stringa `s1` in una collezione di token (parti della stringa) separati dal carattere definito in `s2`.

strcpy

- La funzione `strcpy` copia la stringa `s2` in `s1`.
- Viene restituito il valore di `s1` a seguito della copia.
- `s1` **deve** essere un array in quanto mutabile (modificabile);
- `s2` **può** essere:
  - un array;
  - un puntatore a una stringa;
  - un testo costante (es.: "ciao").

strcpy

es8.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // per usare strcpy, ecc...
#define SIZE 64

int main(void)
{
  // stringa a
  char sa[] = "Ciao"; // oppure char *sa = "Buon"
  char sb[SIZE]; // DEVE essere array per contenere la copia
  char *sc;

  printf("sa = %s\n", sa);

  // copia sa in sb
  strcpy(sb, sa);
  printf("sb = %s\n", sb);

  // oppure
  // sc punterà al primo carattere di sb
  sc = strcpy(sb, "UPO");
  printf("sc = %s\n", sc);

  return 1;
}
```

strcat

- La funzione `strcat` aggiunge il suo secondo argomento (una stringa –costante, puntatore o array) in coda al primo argomento (un array di caratteri). Il primo carattere della seconda stringa sostituisce il carattere di fine stringa `'\0'`.
- Porre attenzione a creare un array di caratteri abbastanza grande da contenere anche i caratteri della seconda stringa.
- È sempre bene inizializzare l'array che conterrà la concatenazione.

strcat

es9.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // per usare strcpy, ecc...
#define SIZE 64

int main(void)
{
  // stringa a
  char sa[] = "Felice"; // oppure char *sa = "Felice"
  char sb[] = "Anno";
  char *sc = "Nuovo";
  char sd[SIZE] = ""; // DEVE essere array per contenere la copia

  strcat(sd, sa);
  strcat(sd, " ");
  strcat(sd, sb);
  strcat(sd, " ");
  strcat(sd, sc);

  printf("Stringa concatenata: %s\n",sd);

  return 1;
}
```

strcmp

- La funzione `strcmp` confronta il suo primo argomento (una stringa) con una seconda stringa.
- Il *formato* delle due stringhe può essere qualsiasi: costante, array o puntatore.
- La funzione restituisce:
  - **0** se le stringhe sono **uguali**;
  - un **valore negativo** se la prima stringa è minore della seconda;
  - un **valore positivo** se la prima stringa è maggiore della seconda.

strcmp

es10.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // per usare strcpy, ecc...
#define SIZE 64

int main(void)
{
  char sa[] = "Happy New Year";
  char *sb = "Happy New Year";
  int r;

  printf("sa: %s\n", sa);
  printf("sb: %s\n", sb);

  r = strcmp(sa,sb);

  if (r==0){
    printf("sa e sb sono uguali (%d)\n", r);
  }
  else{
    printf("sa e sb non sono uguali (%d)\n", r);
  }

  r = strcmp(sa,"Happy New year");

  if (r==0){
    printf("sa e \"Happy New year\" sono uguali (%d)\n", r);
  }
  else{
    printf("sa e \"Happy New year\" non sono uguali (%d)\n", r);
  }

  return 1;
}
```

strchr

- La funzione `strchr` cerca **la prima occorrenza** di un carattere in una stringa.
- Se il carattere viene trovato, la funzione restituisce un puntatore a quel carattere nella stringa, altrimenti `NULL`.
- Il *formato* della stringa può essere qualsiasi: costante, array o puntatore.

strchr

es11.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // per usare strcpy, ecc...

int main(void)
{
  char sa[] = "Happy New Year";
  char *sb = "Buon anno";

  printf("sa: %s\n", sa);
  printf("sb: %s\n", sb);

  char x = 'n';

  // contiene il risultato della ricerca
  char *r = NULL;

  r = strchr(sa, x);

  if (r!=NULL){
    printf("Il carattere '%c' e\' presente in sa all'indirizzo di memoria %p\n", x, r);
  }
  else{
    printf("Il carattere '%c' non e\' presente in sa\n", x);
  }

  r = strchr(sb, x);

  if (r!=NULL){
    printf("Il carattere '%c' e\' presente in sb all'indirizzo di memoria %p\n", x, r);
  }
  else{
    printf("Il carattere '%c' non e\' presente in sb\n", x);
  }

  return 1;
}
```

strstr

- La funzione `strstr` cerca la prima occorrenza di una stringa `s2` in un'altra stringa `s1`.
- Se la stringa viene trovata, la funzione restituisce un puntatore all'inizio di quella parte di stringa, altrimenti `NULL`.
- Il *formato* delle due stringhe può essere qualsiasi: costante, array o puntatore.

strstr

es12.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // per usare strcpy, ecc...

int main(void)
{
  char sa[] = "Happy New Year"; // stringa in formato array
  char *sb = "New";             // stringa in formato puntatore

  printf("sa: %s\n", sa);
  printf("sb: %s\n", sb);

  // contiene il risultato della ricerca
  char *r = NULL;

  // cerca sb in sa
  r = strstr(sa, sb);

  if (r!=NULL){
    printf("La stringa '%s' e\' presente in sa all'indirizzo di memoria %p\n", sb, r);
    printf("Stringa puntata risultante: %s\n", r);
  }
  else{
    printf("La stringa '%s' non e\' presente in sa\n", sb);
  }

  // cerca "new" (stringa costante) in sa
  r = strstr(sa, "new");

  if (r!=NULL){
    printf("La stringa 'new' e\' presente in sa all'indirizzo di memoria %p\n", r);
    printf("Stringa puntata risultante: %s\n", r);
  }
  else{
    printf("La stringa 'new' non e\' presente in sa\n", sb);
  }

  return 1;
}
```

strtok

- La funzione `strtok` viene usata per suddividere una stringa in una serie di **token**.
- Un token è una sequenza di caratteri separata da **delimitatori** (spazi o simboli).
- Ad esempio, nella frase `"Ciao come stai"` si può ritenere lo spazio come un delimitatore e quindi separare la stringa in tre token: `"Ciao"`, `"come"` e `"stai"`.
- Per ottenere questo risultato, la `strtok` inserisce al posto dei delimitatori il carattere `'\0'` e sposta il puntatore alle sotto-stringhe (token) successive.
  - **La stringa iniziale viene quindi modificata**.

strtok

es13.c

```c
int main(void)
{
  // stringa in formato array
  // non può essere puntatore perché viene modificata
  char sa[] = "Happy New Year";
  // il delimitatore può essere un array, un puntatore o una costante
  char *delimitatore = " ";

  printf("Stringa sa: %s\n", sa);

  char *tokenPtr1 = strtok(sa, delimitatore);

  while (tokenPtr1!=NULL)
  {
      printf("Stringa: %s\n", tokenPtr1);
      // la chiamata con puntatore a NULL indica
      // di continuare dall'ultima posizione disponibile del puntatore
      tokenPtr1 = strtok(NULL,delimitatore);
  }

  // sa è stata modificata
  printf("sa dopo la tokenizer: %s \n", sa);

  char sb[] = "Andrea;20;70.88";

  printf("Stringa sb: %s\n", sb);

  char *tokenPtr2 = strtok(sb, ";");

  while (tokenPtr2!=NULL)
  {
      printf("Stringa: %s\n", tokenPtr2);
      tokenPtr2 = strtok(NULL,";");
  }

  // sb è stata modificata
  printf("sb dopo la tokenizer: %s \n", sb);

  return 1;
}
```

FINE PRESENTAZIONE