Stringhe

Una stringa è un vettore di caratteri.

Contiene la sequenza di caratteri che forma la stringa, seguita dal **carattere speciale di fine stringa**: `'\0'`.

Esempio: `char stringa1[10] = {'p', 'i', 'p', 'p', 'o', '\0'};`

Il seguente vettore di caratteri non è una stringa perchè non termina con `'\0'`.

Esempio: `char non_stringa1[2] = {'p', 'i'};`

Inizializzazione di stringhe

Una stringa può anche essere **inizializzata** utilizzando una **stringa letterale**:

Esempio:
```
char stringa2[] = "pippo";     oppure
char stringa2[6] = "pippo";
```

`stringa2` è un array **statico** di 6 caratteri: `'p'`, `'i'`, `'p'`, `'p'`, `'o'`, `'\0'`.

È possibile memorizzare una stringa in un **array di caratteri dinamico**.

Esempio: `char *buffer = malloc(80*sizeof(char));`

In questo caso (come per tutti gli array dinamici) **non** possiamo inizializzare l'array contestualmente alla sua creazione.

Possiamo anche "assegnare" ad una stringa una stringa letterale.

Esempio:
```
char *buffer2;
buffer2 = "pippo";
```

Con questa assegnazione abbiamo assegnato a `buffer2`, di tipo `char*`, la stringa costante `"pippo"`, di tipo `char*` costante.
$\Longrightarrow$ `buffer2` punta al primo carattere della **stringa costante** `"pippo"`.

L'istruzione `buffer2[0] = 't';` **non** dà errore di compilazione.

Dà però **errore in esecuzione** poiché stiamo cercando di cambiare un carattere dichiarato costante.

N.B. questo è diverso da
```
char buffer3[] = "pippo";      che è equivalente a
char buffer3[6] = "pippo";     che è equivalente a
char buffer3[] = {'p', 'i', 'p', 'p', 'o', '\0'};
```
In questo caso la stringa costante `"pippo"` viene usata per inizializzare il vettore `buffer3`.

Inizializzazione di un **vettore di stringhe**:

Esempio: `char *colori[4] = {"rosso", "giallo", "verde", "blu"};`

È un vettore di quattro puntatori a quattro stringhe costanti (di lunghezza $6, 7, 6, 4$).

È equivalente ad inizializzare i quattro puntatori separatamente:

```
char *colori[4];
colori[0] = "rosso";     colori[1] = "giallo";
colori[2] = "verde";     colori[3] = "blu";
```

Ingresso/uscita di stringhe

Stampa di una stringa: si deve utilizzare la specifica di formato "`%s`".

Esempio:
```
printf("%s\n", stringa1);
printf("%s\n", stringa2);
printf("%s\n", buffer2);
```

Vengono stampati tutti i caratteri fino al primo `'\0'` escluso.

Esempio: `printf("%s\n", non_stringa1);` stampa: `pi^D^H"^D^H"^D^A...`

Lettura di una stringa: si deve utilizzare la specifica di formato "`%s`".

Esempio:
```
char buffer[40];
scanf("%s", buffer);
```

1. Vengono letti da input i caratteri in sequenza fino a trovare il primo carattere di spaziatura (spazio, tabulazione, interlinea, ecc.).
2. I caratteri letti vengono messi dentro il vettore `buffer`.
3. Al posto del carattere di spaziatura, viene posto il carattere `'\0'`.

Note:
- il vettore **deve** essere sufficientemente grande da contenere tutti i caratteri letti
- non si usa `&buffer` ma direttamente `buffer` (questo perché `buffer` è di tipo `char*`, ovvero è già un indirizzo)

Manipolazione di stringhe

Per manipolare una stringa si deve accedere ai singoli caratteri singolarmente.

Esempio:
```
for (i = 0; buffer[i] != '\0'; i++) {
  /* fai qualcosa con buffer[i], ad esempio: */
  printf("%c\n", buffer[i]);
}
```

Esempio: Confronto di uguaglianza tra due stringhe.

N.B. **Non** si può usare "`==`" perché questo confronta i puntatori e non le stringhe.
$\Longrightarrow$ Si devono necessariamente scandire le due stringhe.
Implementazione: file `stringhe/strequal.c`

Esercizio: Lunghezza di una stringa.
Implementazione: file `stringhe/strlungh.c`

In realtà queste funzioni, come molte altre, sono disponibili nella libreria sulle stringhe.

Caratteri e stringhe

Caratteri e stringhe sono diversi e **non** vanno confusi:

- un carattere è in realtà un intero
  per denotare una costante di tipo carattere: `'x'`
- una stringa è un vettore di caratteri che termina con il carattere `'\0'`
  per denotare costanti di tipo stringa: `"un esempio di stringa"`
- una variabile di tipo stringa è in realtà un puntatore al primo carattere del vettore

Esempio:

`char c = 'a';`	carattere
`char *s = "a";`	puntatore alla stringa costante `"a"`
`char v[] = "a";`	vettore di 2 caratteri inizializzato a `{'a', '\0'}`

```
printf("%d %d %d\n", sizeof(c), sizeof(s), sizeof(v));
```

stampa:
```
1 4 2
```


Funzioni di libreria per la gestione dei caratteri

Per usare le funzioni per la gestione dei caratteri **è necessario**: `#include <ctype.h>`

Tutte le funzioni operano sui caratteri e su `EOF`.
Però: `char` può essere `unsigned`, mentre `EOF` normalmente è -1.

$\Longrightarrow$ Tutte le funzioni hanno parametri di tipo `int` (e non di tipo `char`).
Però possono prendere argomenti di tipo `char`, poiché `char` viene promosso ad `int`..

Funzioni di conversione maiuscolo - minuscolo

`int tolower(int c);`	se `c` è una lettera maiuscola, la converte in minuscolo altrimenti restituisce `c` invariata
`int toupper(int c);`	se `c` è una lettera minuscola, la converte in maiuscolo altrimenti restituisce `c` invariata

Funzioni che determinano il tipo di un carattere

- restituiscono vero (1) se il carattere è tra quelli indicati
- restituiscono falso (0) altrimenti

`int isdigit(int c);`	cifra
`int isxdigit(int c);`	cifra esadecimale
`int isalpha(int c);`	lettera alfabetica (minuscola o maiuscola)
`int isalnum(int c);`	cifra o lettera alfabetica
`int islower(int c);`	lettera alfabetica minuscola
`int isupper(int c);`	lettera alfabetica maiuscola
`int isspace(int c);`	carattere di spazio bianco (`' '`, `'\n'`, `'\r'`, `'\t'`, `'\v'`, `'\f'`)
`int iscntrl(int c);`	carattere di controllo (tra 0 e $0x1F$ più $0x7F$ = DEL) (`'\t'`, `'\v'`, `'\a'`, `'\b'`, `'\r'`, `'\n'`, ...)
`int isprint(int c);`	carattere stampabile (incluso lo spazio)
`int isgraph(int c);`	carattere stampabile diverso dallo spazio
`int ispunct(int c);`	carattere stampabile non alfanumerico o di spazio (ovvero un carattere di punteggiatura)

Funzioni di libreria per la conversione di stringhe

È necessario: `#include <stdlib.h>`

Convertono le strighe formate da cifre in valori interi ed in virgola mobile.

Funzioni `atoi`, `atol`, `atof`

```
int atoi(const char *str);
```

- converte la stringa puntata da `str` in un `int`
- la stringa deve contenere un numero intero valido
- in caso contrario il risultato è indefinito
- spazi bianchi iniziali vengono ignorati
- il numero può essere concluso da un qualsiasi carattere che non è valido in un numero intero (spazio, lettera, virgola, punto, . . .)

Esempio: file `stringhe/strtonum.c`

`atoi("123")`	restituisce l'intero 123
`atoi("123.45")`	restituisce l'intero 123 (`".45"` viene ignorato)
`atoi("  123.45")`	restituisce l'intero 123 (`"  "` e `".45"` ignorati)


```
long atol(const char *str);
```

- analoga ad `atoi`, solo che restituisce un `long int`

```
double atof(const char *str);
```

- analoga ad `atoi` ed `atol`, solo che restituisce un `double`
- il numero può essere concluso da un qualsiasi carattere non valido in un numero in virgola mobile

Esempio: file `stringhe/strtonum.c`

`atof("123.45")`	restituisce 123.45
`atof("1.23xx")`	restituisce 1.23
`atof("1.23.45")`	restituisce 1.23

Funzioni `strtod`, `strtol`, `strtoul`

```
double strtod(const char *inizio, char **fine);
```

- converte la stringa puntata da `inizio` in un `double`
- `*fine` viene fatto puntare al primo carattere successivo alla porzione di stringa che è stata convertita
- se il secondo argomento è `NULL`, allora viene ignorato
- se la conversione non è possibile, allora `*fine` viene posto uguale ad `inizio` e viene restituito 0

Esempio: file `stringhe/strtonum.c`

```
double d;
char *stringa = "123.45Euro";
char *resto;

d = strtod(stringa, &resto);
printf("%g\n", d);
printf("%s\n", resto);
```

Stampa

```
123.45
Euro
```

```
long strtol(const char *inizio, char **fine, int base);
```

- converte la stringa puntata da `inizio` in un `long`
- la stringa deve contenere un numero nella `base` specificata
- la `base` deve essere tra 2 e 36, oppure 0:
  - se è 0, la base da usare è determinata in base alla notazione per le costanti (ottale, decimale, esadecimale)
  - se è > 10, le cifre tra 10 e 35 sono rappresentate dalle lettere A–Z (oppure a–z)
- per il resto come `strtod`

```
unsigned long strtoul(const char *inizio, char **fine, int base);
```

- analoga a `strtol`, solo che restituisce un `unsigned long`

Esempio: file `stringhe/strtonum.c`

```
long i;
char *resto;

i = strtol("12000lire", &resto, 10);
printf("%ld\n", i);
printf("%s\n", resto);

i = strtol("FFGFF", &resto, 16);
printf("%ld\n", i);
printf("%s\n", resto);

i = strtol("0xFFGFF", &resto, 0);
printf("%ld\n", i);
printf("%s\n", resto);
```

Stampa

```
12000
lire

255
GFF

255
GFF
```

Esempio: Conversione di tutti i numeri in una stringa e memorizzazione in un vettore.

Implementazione: file `stringhe/strnums.c`

Funzioni di libreria per l'input/output di stringhe e caratteri

È necessario: `#include <stdio.h>`

Input e output di caratteri

```
int getchar(void);
```

- legge il prossimo carattere da standard input e lo restituisce

```
int putchar(int c);
```

- manda `c` in standard output
- restituisce `EOF` (-1) se si verifica un errore

Input e output di stringhe

`char *gets(char *str);`

- legge i caratteri da standard input e li inserisce nel vettore `str`
- la lettura termina con un `'\n'` o un `EOF`, che **non** viene inserito nel vettore
- la stringa nel vettore viene terminata con `'\0'`
- **Attenzione:** non c'è alcun modo di limitare la lunghezza della sequenza immessa
  $\Longrightarrow$ vettore allocato potrebbe non bastare

`int puts(const char *str);`

- manda la stringa `str` in standard output (inserisce un `'\n'` alla fine)

Input e output di stringhe da/su stringhe

`int sprintf(char *str, const char *format, ...);`

- come `printf`, solo che invece di scrivere su standard output, scrive nel vettore `str`

`int sscanf(char *str, const char *format, ...);`

- come `scanf`, solo che invece di leggere da standard input, legge dalla stringa `str`

Funzioni di libreria per la manipolazione di stringhe

È necessario: `#include <string.h>`

Funzioni di copia

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

- copia la stringa `src` nel vettore `dest`
- restituisce il valore di `dest`
- `dest` dovrebbe essere sufficientemente grande da contenere `src`
- se `src` e `dest` si sovrappongono, il comportamento di `strcpy` è indefinito

Esempio:

```
char a[10], b[10];
char *x = "Ciao";
char *y = "mondo";
strcpy(a, x);   strcpy(b, y);
puts(a);        puts(b);
```

Stampa:

```
Ciao
mondo
```


`char *strncpy(char *dest, const char *src, size_t n);`

- copia un massimo di `n` caratteri della stringa `src` nel vettore `dest` (`size_t` è il tipo del valore restituito da `sizeof`, ovvero `unsigned long` o `unsigned int`)
- restituisce il valore di dest
- **Attenzione:** il carattere `'\0'` finale di `src` viene copiato **solo** se `n` è $\geq$ della lunghezza di `src+1`

Esempio:

```
char a[10], b[10];
char *x = "Ciao";
char *y = "mondo";

strncpy(a, x, 5);
puts(a);                    stampa: Ciao
strncpy(b, y, 5);           b non è terminata da '\0'
b[5] = '0';
puts(b);                    stampa: mondo
```

Funzioni di concatenazione

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

- accoda la stringa `src` a quella nel vettore `dest` (il primo carattere di `src` si sostituisce al `'\0'` di `dest`) e termina `dest` con `'\0'`
- restituisce il valore di `dest`
- `dest` dovrebbe essere sufficientemente grande da contenere tutti i caratteri di `dest`, di `src`, ed il `'\0'` finale
- se `src` e `dest` si sovrappongono, il comportamento di `strcat` è indefinito

Esempio:

```
char a[10], b[10];
char *x = "Ciao",  *y = "mondo";
strcpy(a, x);
strcat(a, y);
puts(a);
```

stampa: `Ciaomondo`

```
char *strncat(char *dest, const char *src, size_t n);
```

- accoda al più `n` caratteri di `src` alla stringa nel vettore `dest` (il primo carattere di `src` si sostituisce al `'\0'` di `dest`)
- termina in ogni caso `dest` con `'\0'`

Funzioni di confronto

```
int strcmp(const char *s1, const char *s2);
```

- confronta le stringhe `s1` ed `s2`
- restituisce:
  - 0 se `s1` $=$ `s2`
  - un valore < 0 se `s1` $<$ `s2`
  - un valore > 0 se `s1` $>$ `s2`
- il confronto è quello lessicografico: i caratteri vengono confrontati uno ad uno, ed il primo carattere diverso (o la fine di una delle due stringhe) determina il risultato
- per il confronto tra due caratteri viene usato il codice

Esempio:

```
char *s1 = "abc";
char *s2 = "abx";
char *s3 = "abc altro";

printf("%d\n", strcmp(s1, s1));
printf("%d\n", strcmp(s1, s2));
printf("%d\n", strcmp(s1, s3));
printf("%d\n", strcmp(s2, s1));
```

Stampa:

```
0
-1
-1
1
```


Attenzione: per verificare l'uguaglianza bisogna confrontare il risultato con 0

Esempio:

```
if (strcmp(s1, s2) == 0)
  printf("uguali\n");
else
  printf("diverse\n");
```

```
int strncmp(const char *s1, const char *s2, size_t n);
```

- confronta al più `n` caratteri di `s1` ed `s2`

Esempio:

```
char *s1 = "abc";
char *s3 = "abc altro";

printf("%d\n", strncmp(s1, s3, 3));
```

Stampa `0`, ovvero le due stringhe sono uguali.

Altre funzioni di libreria sulle stringhe

- lunghezza di una stringa (pari al numero di caratteri che precedono `'\0'`)
  `size_t strlen(const char *str);`
- di ricerca (di caratteri, di sottostringhe, ...)
- di manipolazione della memoria

Esercizio: Fornire un'implementazione delle funzioni di libreria su caratteri e stringhe.

Esercizio: Programma che legge da tastiera quattro stringhe che rappresentano interi, le converte in interi, le somma, e stampa la somma.

Esercizio: Programma che legge una data nel formato "`12/05/2002`" e la converte nel formato "`12 maggio 2002`".
Implementazione: file `stringhe/convdata.c`

Esercizio: Programma che legge un numero tra 0 e $999\,999$ e lo stampa in lettere.

Esercizio: Modificare il programma per la gestione del deposito di autobus tenendo conto che ogni autobus è identificato da un nome invece che da un codice numerico.

Programmi con argomenti passati da linea di comando

Quando compiliamo un programma viene generato un file eseguibile (estensione `.EXE` in Windows). I programmi possono prendere degli argomenti dalla riga di comando con la quale vengono eseguiti.
Esempio: `copy file1 file2`

I programmi visti finora non prendevano argomenti. La funzione `main` non prendeva parametri: `int main(void) { ... }`

Per poter usare gli argomenti con cui il programma è stato lanciato, la definizione di `main` deve diventare:

```
int main(int argc, char *argv[]) { ... }
```

- `argc` ... numero di argomenti con cui il programma è stato lanciato più 1
- `argv` ... vettore di `argc` stringhe che compongono la riga di comando
  `argv[0]` è il nome del programma stesso
  `argv[1]`, ..., `argv[argc-1]` sono gli argomenti (separati da spazio)
- ogni `argv[i]` è una stringa (terminata da `'\0'`)
- `argc` e `argv` vengono inizializzati dal sistema operativo (le stringhe di `argv` sono allocate dinamicamente)

Esempio: Stampa del numero di argomenti ricevuti da linea di comando.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  printf("Ho ricevuto %d argomenti\n", argc-1);
  return 0;
}
```

Compilando si ottiene `argnum.exe`, che può essere eseguito:

```
> argnum primo secondo terzo
> Ho ricevuto 3 argomenti
```

Esempio: Stampa degli argomenti ricevuti da linea di comando.
Implementazione: file `stringhe/argprint.c`

Esempio: Stampa della somma di due numeri interi passati come argomenti.

N.B. I due numeri sono passati come stringhe e non come numeri.
$\Longrightarrow$ Vanno convertiti in numeri usando `atoi`.
Implementazione: file `stringhe/argsum.c`

Esempio: Eliminazione dell'estenzione dal nome di un file passato come argomento.

- il nome di un file è una stringa che può contenere un `'.'`
- vogliamo mantenere solo la parte che precede il `'.'` e memorizzarla in una nuova stringa
- il programma deve venire lanciato con un singolo argomento, che è la stringa da cui togliere l'estensione: possiamo accedere alla stringa con `argv[1]`
  i caratteri della stringa sono `argv[1][i]`
- per poter eliminare l'estensione, scandiamo la stringa
  - usiamo un indice `i` per la scansione
  - ci fermiamo quando incontriamo `'.'` oppure `'\0'`
    $\Longrightarrow$ la condizione di terminazione del ciclo deve essere:
    `argv[1][i] != '\0' && argv[1][i] != '.'`
- per memorizzare la stringa senza estensione usiamo un array dinamico

Implementazione: file `stringhe/argnoext.c`

Esercizio: Modificare i programmi visti a lezione che leggono dati da input, in modo che i dati vengano passati come argomenti al programma invece che letti da input.