# LABORATORIO DI PROGRAMMAZIONE 1
## CORSO DI LAUREA IN MATEMATICA
## UNIVERSITÀ DEGLI STUDI DI MILANO
### 2019–2020

Indice



Ultima revisione: 16 febbraio 2020.

**Avvertenza**

In questa lezione di laboratorio farete esercizi sull'uso dei puntatori a carattere per la manipolazione delle stringhe. Per la lettura delle stringhe inserite dall'utente userete sempre la funzione `fgets` della libreria standard. L'Esercizio 1 riporta il codice necessario per la lettura di una stringa con `fgets`. Se preferite, potete anche scrivere all'inizio dell'esercitazione una funzione ausiliaria che usi `fgets` per acquisire una stringa da tastiera, e poi riutilizzarla nei diversi esercizi. Nel caso vogliate acquisire un carattere inserito dall'utente, usate la funzione `getchar()` della libreria standard.

mainrev.c

```c
#include <stdio.h>

int main(void)
{
        void reverse(char*, char*);

        /* BUFSIZ e' una costante definita
        in stdio.h */
        char str1[BUFSIZ], str2[BUFSIZ];

        printf("Inserisci una stringa: ");
        if( fgets (str1, BUFSIZ, stdin)==NULL )
        {
                printf("Errore in lettura!\n");
                return -1;
        }
        /* Sovrascrive \n finale */
        int i=0;
        for (;str1[i]!='\n';i++);
        str1[i]='\0';

        reverse(str1, str2);

        printf("La stringa al contrario e':\n%s\n", str2);

        return 0;
}
```

FIGURA 1. Esempio di implementazione della funzione `main` per l'Esercizio 1.

## ESERCIZIO 1

*Funzione* `reverse`.

Tempo: 15 min.

Scrivete un programma che chieda all'utente di inserire una stringa, invochi una funzione che restituisce una copia della stringa scritta in ordine inverso, la visualizzi

e termini. La funzione che inverte la stringa ha prototipo

```
void reverse(char *s, char *t)
```

dove il primo parametro rappresenta la stringa da invertire, e il secondo la stringa invertita. In `main`, invocate la funzione passandole come primo parametro (il nome di un) array di tipo `char` inizializzato con una costante stringa — ossia la stringa letta dall'utente — e come secondo parametro (il nome di un) array di tipo `char` della medesima dimensione del primo. Un esempio di come potrebbe essere scritta la funzione `main` del programma è in Fig. 1.

Prestate attenzione al fatto che la funzione `fgets` include nell'array di caratteri letti anche il carattere `'\n'` di terminazione riga: nell'esempio è incluso il codice necessario a sovrascrivere `'\n'` con `'\0'`.

## Esercizio 2

*Funzione* `equals`.

Tempo: 15 min.

Scrivete una funzione di prototipo

```
int equals(char *s, char *t)
```

che accetti in ingresso due stringhe, e restituisca in uscita un intero di valore non negativo nel caso le due stringhe siano uguali, e nullo altrimenti. Scrivete una funzione `main` atta a testare adeguatamente la vostra funzione.

Come si comporta la vostra implementazione di `equals` se uno o entrambi i puntatori in ingresso valgono `NULL`?

## Esercizio 3

*Funzione* `palindroma`.

Tempo: 30 min.

Scrivete una funzione di prototipo

```
int palindroma(char *s)
```

che accetti in ingresso una stringa, e restituisca in uscita un intero di valore non negativo nel caso la stringa sia palindroma, e nullo altrimenti. Scrivete una prima versione che utilizzi le funzioni `reverse` e `equals` degli Esercizi 1 e 2. Scrivete una funzione `main` atta a testare adeguatamente la vostra funzione. Scrivete poi una seconda implementazione della funzione `palindroma` che utilizzi solo la stringa `s` passata in argomento, senza invocare funzioni ausiliarie e senza allocare memoria per una copia di `s`.

Questa seconda implementazione è meno facile da scrivere, e può sembrare una soluzione meno elegante rispetto alla prima: osserverete, tuttavia, che essa è senz'altro la più efficiente delle due dal punto di vista dell'uso delle risorse fisiche a disposizione (spazio in memoria, tempo di calcolo).

Come si comporta la vostra implementazione di `palindroma` se il puntatore in ingresso valgono `NULL`?

## Esercizio 4

*Confronto lessicografico tra stringhe*.

Tempo: 25 min.

Scrivete una funzione che permetta di confrontare le stringhe secondo l'ordinamento lessicografico (cioè quello dei dizionari). Il prototipo è:

```
int lex(char *s, char *t)
```

I parametri in ingresso rappresentano le due stringhe da confrontare. La funzione restituisce un intero positivo se `s` precede strettamente `t` nell'ordinamento, un intero negativo se `t` precede strettamente `t` nell'ordinamento, e zero nel caso in cui le due stringhe siano uguali. Scrivete una procedura `main` appropriata che permetta di testare la vostra implementazione.

## Esercizio 5

*Funzione* `clean`**.**

Tempo: 20 min.

Scrivete un programma che chieda all'utente di inserire una stringa, invochi una funzione che restituisce una copia della stringa privata degli spazi, la visualizzi e termini. La funzione che elimina gli spazi dalla stringa ha prototipo

```
void clean(char *s, char *t)
```

dove il primo parametro rappresenta la stringa da ripulire, e il secondo la stringa ripulita. In `main`, invocate la funzione passandole come primo parametro (il nome di un) array di tipo `char` inizializzato con una costante stringa, e come secondo parametro (il nome di un) array di tipo `char` della medesima dimensione del primo.

## Esercizio 6

*Funzione* `clean` *parametrica***.**

Tempo: 10 min.

Riutilizzate il codice che avete scritto per l'Esercizio 5.

Scrivete un programma che chieda all'utente di inserire una stringa, invochi una funzione che restituisce una copia della stringa privata di tutte le occorrenze di un carattere passato in argomento alla funzione, la visualizzi e termini. La funzione che elimina le occorrenze del carattere dalla stringa ha prototipo

```
void clean(char *s, char *t, char c)
```

dove il primo parametro rappresenta la stringa da ripulire, il secondo la stringa ripulita, e il terzo il carattere le cui occorrenze sono da eliminare. In `main`, invocate la funzione passandole come primo parametro (il nome di un) array di tipo `char` inizializzato con una costante stringa, e come secondo parametro (il nome di un) array di tipo `char` della medesima dimensione del primo.

## Esercizio 7

*Funzione* `clean` *parametrica, con più caratteri***.**

Tempo: 25 min.

Scrivete un programma che chieda all'utente di inserire una stringa, invochi una funzione che restituisce una copia della stringa privata di tutte le occorrenze degli elementi di un insieme di caratteri passato in argomento alla funzione, la visualizzi e termini. La funzione che elimina le occorrenze dei caratteri dalla stringa ha prototipo

Riutilizzate il codice che avete scritto per l'Esercizio 6.

```
void clean(char *s, char *t, char *u)
```

dove il primo parametro rappresenta la stringa da ripulire, il secondo la stringa ripulita, e il terzo una stringa i cui caratteri sono quelli le cui occorrenze sono da eliminare (escluso il `\0` di terminazione di `u`). Per esempio, se `u` punta a `"A x"`, dalla stringa `s` saranno eliminate tutte le occorrenze dei caratteri `'A'`, `' '` (=spazio) e `'x'`. Nel `main`, invocate la funzione passandole come primo parametro (il nome di un) array di tipo `char` inizializzato con una costante stringa, come secondo parametro (il nome di un) array di tipo `char` della medesima dimensione del primo e come terzo parametro (il nome di un) array di tipo `char` inizializzato con una costante stringa.

Dipartimento di Matematica *Federigo Enriques*, Università degli Studi di Milano, via Cesare Saldini, 50, I-20133 Milano