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

INDICE

Parte 1. Input/Output standard da file 2
Esercizio 1 2
Eco di file 2
Tempo: 15 min. 2
Esercizio 2 2
Eco di file parametrico 2
Tempo: 10 min. 2
Esercizio 3 2
Copia di file 2
Tempo: 25 min. 2
Esercizio 4 2
Copia di `stdin` *su file* 2
Tempo: 20 min. 2

Parte 2. Array di puntatori 3
Esercizio 5 3
`Reverse` *dalla riga di comando* 3
Tempo: 15 min. 3
Esercizio 6 3
`Clean` *dalla riga di comando* 3
Tempo: 20 min. 3
Esercizio 7 3
Settimana anglofona 3
Tempo: 30 min. 4
Esercizio 8 4
Ordinamento delle stringhe per lunghezza 4
Tempo: 40 min. 4
Esercizio 9 4
Ordinamento lessicografico delle stringhe 4
Tempo: 20 min. 4

Ultima revisione: 20 febbraio 2020.

Parte 1. Input/Output standard da file

Esercizio 1

Eco di file.

Tempo: 15 min.

Si scriva un programma che visualizzi il suo stesso codice sorgente sulla console, e termini.

Suggerimento. Il programma apre il file sorgente in lettura con `fopen`, e legge il contenuto del file una riga per volta con `fgets`.

Esercizio 2

Eco di file parametrico.

Tempo: 10 min.

Si modifichi il programma dell'Esercizio 1 di modo che esso riceva come argomento dalla riga di comando un nome di file, e visualizzi il contenuto del file sulla console. Si gestiscano gli errori: cosa succede se il nome non corrisponde a un file esistente? E se manca l'argomento sulla riga di comando?

Esercizio 3

Copia di file.

Tempo: 25 min.

Si scriva un programma che riceva due nomi di file dalla riga di comando, e che copi il primo file sul secondo (carattere per carattere). Se il primo file non può essere aperto, visualizzate un messaggio appropriato. Cosa succede se il file di destinazione della copia è già esistente e contiene dei dati? Migliorate il programma facendo sì che esso chieda all'utente conferma di voler sovrascrivere il file di destinazione prima di procedere.

Questo vuol dire: usate la funzione `getc` per leggere da file, non la funzione `fgets`.

Suggerimento. Per controllare se il file di destinazione esiste, potete tentare di aprirlo in lettura e testare il valore restituito da `fopen`.

Esercizio 4

Copia di `stdin` *su file.*

Tempo: 20 min.

Si scriva un programma che riceva un nome di file dalla riga di comando, e che poi copi (riga per riga) tutto quanto l'utente digita da tastiera sul file. Il programma termina quando l'utente invia il segnale di `EOF` da tastiera, ossia quando digita CTRL+D (sistemi Unix/Linux/OS X) oppure CRTRL+Z (sistemi Windows). Si noti che se invece l'utente digita invio, sul file occorre scrivere una riga vuota costituita dal solo ritorno a capo. Gestite gli errori.

Questo vuol dire: usate la funzione `fgets` per leggere da file, non la funzione `getc`.

Parte 2. Array di puntatori

Esercizio 5

`Reverse` *dalla riga di comando*.

Tempo: 15 min.

Modificate la vostra soluzione all'Esercizio 1 della Lezione 6 in modo che l'utente possa inserire la stringa di cui calcolare la stringa riflessa dalla riga di comando. Per esempio, se avete chiamato il codice eseguibile del vostro programma `rev`, allora digitando dalla shell

```
./rev Ailatiditalia
```

dovreste ottenere in uscita:

```
ailatiditaliA
```

Osservate che non è necessario memorizzare la stringa riflessa — basta visualizzarla.

Modificate la vostra soluzione all'esercizio in modo che il programma visualizzi un messaggio d'errore appropriato nel caso un cui non vi sia l'argomento necessario sulla riga di comando.

Esercizio 6

`Clean` *dalla riga di comando*.

Tempo: 20 min.

Modificate la vostra soluzione all'Esercizio 5 della Lezione 6 in modo che l'utente possa inserire la stringa da ripulire e la stringa che indica i caratteri da eliminare dalla riga di comando. Per esempio, se avete chiamato il codice eseguibile del vostro programma `clean`, allora digitando dalla shell

```
./clean Teleologicamente eTo
```

dovreste ottenere in uscita:

```
llgicamnt
```

Osservate che, in `main`, per poter dimensionare correttamente l'array che conterrà la stringa ripulita, dovrete calcolare la lunghezza della stringa in ingresso dalla riga di comando. Provate anche a scrivere una versione del programma che non memorizza la stringa ripulita, ma stampa semplicemente il risultato desiderato.

Modificate la vostra soluzione all'esercizio in modo che il programma visualizzi messaggi d'errore appropriati nel caso un cui vi siano troppi argomenti sulla riga di comando, o ve ne siano troppo pochi.

Esercizio 7

Settimana anglofona.

Tempo: 30 min.

Scrivete un programma che accetti in ingresso dalla riga di comando il nome di un giorno della settimana, e produca in uscita il corrispondente nome in inglese.

A partire da Lunedì, i giorni della settimana in inglese sono: `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday` e `Sunday`.

Il programma userà una funzione di prototipo

```
int indice(char *s)
```

che accetta in ingresso una stringa, e restituisce l'indice corrispondente del giorno della settimana, da 0 a 6, oppure -1 se la stringa non è interpretabile come giorno della settimana. La funzione deve comportarsi in modo da non far differenza fra maiuscole e minuscole: se il parametro in ingresso è `lUnedi`, il valore restituito è 0. Trascurate per semplicità gli accenti finali: l'utente dovrà scrivere, per esempio, `Lunedi` e non `Lunedì` o `Lunedi'`. Il programma userà poi un array di puntatori a carattere per produrre in uscita il nome del giorno specificato dall'utente, in inglese. Gestite gli errori in modo appropriato.

Suggerimenti. Convertite l'intera stringa passata in argomento in maiuscola, o in minuscola, usando le funzioni `toupper()` o `tolower()` di `ctype.h`. Poi confrontate la stringa con gli elementi di un opportuno array di stringhe, inizializzato coi nomi dei giorni della settimana in maiuscola, o in minuscola. Usate la funzione

`int strcmp(char *, char *)`

di `string.h`, che restituisce 0 se, e solo se, le due stringhe in ingresso sono uguali.

Esercizio 8

Ordinamento delle stringhe per lunghezza.

Tempo: 40 min.

Scrivete un programma che chieda all'utente di inserire 5 stringhe, e le memorizzi in un array di puntatori a `char` appropriatamente dimensionato. Il programma chiama poi una funzione di prototipo

```
void ordina(char **s, int lung)
```

il cui primo parametro rappresenta l'array di puntatori a `char`, e il cui secondo parametro rappresenta il numero di stringhe puntate dagli elementi dell'array (in questo esempio, 5). La funzione ordina l'array di stringhe in ordine non decrescente di lunghezza. Scrivete una funzione ausiliaria per il calcolo della lunghezza di una stringa. Come algoritmo di ordinamento usate Bubble Sort, che avete implementato nell'Esercizio 11 della Lezione 4.

Esercizio 9

Ordinamento lessicografico delle stringhe.

Tempo: 20 min.

Modificate la soluzione dell'Esercizio 8 di modo che le stringhe inserite dall'utente siano ordinate secondo l'ordine lessicografico, invece che secondo la loro lunghezza. Per il confronto lessicografico fra stringhe, riutilizzate la funzione che avete scritto per risolvere l'Esercizio 4 della Lezione 6.