Capitolo 8

Lezione 19 aprile 2019
Esercitazione 4

8.1 Modalità di consegna e valutazione esercitazioni

V. Esercitazione 1, 2.

Date di assegnazione e consegna esercitazione 4 19 aprile → 1 maggio

Note aggiuntive Le funzioni consegnate non devono produrre nessun output su terminale. Nel caso siano presenti istruzioni che producono output (es. utilizzando printf), toglietele/commentatele prima di consegnare poiché possono allungare il tempo di esecuzione ed interferire con il processo di correzione. Il programma deve poter essere compilato senza errori o warning con le opzioni `-g3 -fsanitize=address -fsanitize=undefined -std=gnu89 -pedantic-errors -Wall -Wextra` ed essere eseguito senza errori.
IMPORTANTE: la memoria allocata dinamicamente deve essere rilasciata prima del termine dell'esecuzione. Si raccomanda di seguire il corretto ordine di rilascio della memoria allocata per prevenire effetti indesiderati, ad esempio rilasciando prima gli elementi di un array frastagliato e poi l'array stesso.

Valutazione Come per le esercitazioni precedenti ciascun esercizio corretto (compila senza errori o warning, non causa errori di esecuzione, ritorna il valore corretto per tutti i test case) viene valutato 1, quindi l'esercitazione complessivamente sarà valutata tra 0 e 5 punti. La soglia per poter accedere alla valutazione del progetto rimane fissata a 9 punti totali per le 5 esercitazioni.

8.2 Esercitazione 4 - Esercizio A

Scrivere una funzione C che data una stringa contenente un nome di file, conti il numero di caratteri, parole, linee contenute nel file indicato. Una parola è una sottosequenza massimale non contenente spazi, quindi non estensibile senza includere spazi (es: "aa b bb.c" contiene le parole "aa", "b" e "bb.c" ma non la parola "bb"). Due linee sono separate dal carattere LF ('\n'), dal carattere CR ('\r') oppure da entrambi CRLF oppure (LFCR).
Si supponga che il file contenga solamente caratteri nel range 32-126 oltre CR e LF.

Attenzione:

- **la punteggiatura, senza spazi, non interrompe le parole (es: S.r.l. è una parola, non tre)**
- **devono essere contate anche le linee vuote o composte di soli spazi**
- **nel conteggio del numero di caratteri devono essere considerati anche spazi, LF e CR. I caratteri CR ed LF devono in tutti i casi essere conteggiati separatamente come caratteri (CRLF: due caratteri)**

- **un file vuoto contiene 0 linee, 0 parole, 0 caratteri**
- **un file con una sola linea vuota contiene 1 linea, 0 parole, 1 o 2 caratteri (a seconda che il file di input contenga LF oppure CRLF)**
- **il comando wc può essere utilizzato per verificare le vostre soluzioni, ma fate attenzione che conta il numero di newline, non il numero di linee come richiesto da questo esercizio. I due valori sono diversi nel caso l'ultima linea non sia terminata da un newline.**

La funzione deve utilizzare dei puntatori passati come parametri per restituire i risultati al chiamante e ritornare il valore 1 nel caso abbia completato la lettura del file oppure 0 in caso di errori di apertura del file (es: file non esistente).

È consentito utilizzare le funzioni in stdlib.h, string.h e stdio.h. Si consiglia di utilizzare la funzione getline per leggere le linee del file.

Consegna Il prototipo della funzione **DEVE** essere

```
int wc(const char *filename, int *chars, int *words, int *lines);
```

Esempio Un file contenente il seguente testo

```
linea 1.1
linea  due
```

contiene 4 parole, 2 linee e 20 caratteri (9 caratteri, LF, 10 caratteri).

8.3 Esercitazione 4 - Esercizio B

Scrivere una funzione C che date due stringhe contenenti i nomi dei file di input/output ed una stringa di caratteri da eliminare, crei un file (con il nome indicato per l'output) il cui contenuto sia uguale al contenuto del file di input ad eccezione dei caratteri contenuti nella stringa che devono essere rimossi. **La funzione deve ritornare il valore 1 nel caso abbia completato la lettura del file oppure 0 in caso di errori di apertura del file (es: file non esistente).** È consentito utilizzare le funzioni in stdlib.h, string.h e stdio.h.

Consegna Il prototipo della funzione **DEVE** essere

```
int elimina_caratteri(const char *input, const char *output, const char *elimina);
```

Esempio Dato il file prova.txt contenente il seguente testo

```
linea 1.1
linea due
```

dopo l'esecuzione di

```
int elimina_caratteri("prova.txt", "risultato.txt", "ei");
```

il file risultato.txt conterrà:

```
lna 1.1
lna du
```

8.4 Esercitazione 4 - Esercizio C

Scrivere una funzione C che date due stringhe contenenti i nomi dei file di input/output, scriva nel file di output le parole (v. definizione esercizio A) contenute nel file di input scrivendone una per riga, senza spazi e senza ripetizioni (ciascuna parola deve apparire una sola volta nel risultato). **L'ordine delle parole nel file risultato non è rilevante.**

La funzione deve ritornare il valore 1 nel caso abbia completato la lettura del file oppure 0 in caso di errori di apertura del file (es: file non esistente).

È consentito utilizzare le funzioni in stdlib.h, string.h e stdio.h (incluse bsearch e qsort).

Consegna Il prototipo della funzione **DEVE** essere

```
int lista_parole(const char *input, const char *output);
```

Esempio Dato il file prova.txt contenente il seguente testo

```
la funzione deve elencare le parole contenute in un file.
le parole non devono essere indicate più di una volta.
```

dopo l'esecuzione di

```
int lista_parole("prova.txt", "risultato.txt");
```

il file risultato.txt conterrà:

```
contenute
deve
devono
di
elencare
essere
file.
funzione
in
indicate
la
le
non
parole
più
un
una
volta.
```

8.5 Esercitazione 4 - Esercizio D

Scrivere una funzione C che date due stringhe contenenti i nomi dei file di input/output, scriva nel file di output le parole (v. definizione esercizio A) contenute nel file di input (una per riga, senza spazi e senza ripetizioni) e il numero di loro occorrenze. Ciascuna linea del risultato dovrà contenere nell'ordine: il numero di occorrenze di una parola, uno spazio, la parola stessa, il carattere di fine linea (es: "5 cane.\n"). **L'ordine delle parole nel file risultato non è rilevante. La funzione deve ritornare il valore 1 nel caso abbia completato la lettura del file oppure 0 in caso di errori di apertura del file (es: file non esistente).** È consentito utilizzare le funzioni in stdlib.h, string.h e stdio.h (incluse bsearch e qsort).

Consegna Il prototipo della funzione **DEVE** essere

```
int conta_parole(const char *input, const char *output);
```

Esempio Dato il file prova.txt contenente il seguente testo

```
la funzione deve elencare le parole contenute in un file.
le parole non devono essere indicate più di una volta.
```

dopo l'esecuzione di

```
int lista_parole("prova.txt", "risultato.txt");
```

il file risultato.txt conterrà:

```
1 contenute
1 deve
1 devono
1 di
1 elencare
1 essere
```

```
1 file.
1 funzione
1 in
1 indicate
1 la
2 le
1 non
2 parole
1 più
1 un
1 una
1 volta.
```

8.6 Esercitazione 4 - Esercizio E

Scrivere una funzione C che date tre stringhe contenenti i nomi di due file di input ed di un file di output, scriva nel file di output le parole (v. definizione esercizio A) contenute sia nel primo file di input che nel secondo (una per riga, senza spazi e senza ripetizioni). L'ordine delle parole non è rilevante.

La funzione deve ritornare il valore 1 nel caso abbia completato la lettura del file oppure 0 in caso di errori di apertura del file (es: file non esistente). È consentito utilizzare le funzioni in stdlib.h, string.h e stdio.h (incluse bsearch e qsort).

Consegna Il prototipo della funzione **DEVE** essere

```
int intersezione_parole(const char *input1, const char *input2, const char *output);
```

Esempio Dati il file prova1.txt contenente il seguente testo

```
file numero uno
trovare l'intersezione
```

ed il file prova2.txt contenente il seguente testo

```
file numero due
intersezione
```

dopo l'esecuzione di

```
int intersezione_parole("prova1.txt", "prova2.txt", "risultato.txt");
```

il file risultato.txt conterrà:

```
file
numero
```