Scheda di lavoro N. 10

Questa è la *scheda dei files*. Si tratta di esercizi sulla lettura e scrittura di dati da e su file. Per eseguire questi esercizi dovrai aver assimilato il contenuto del capitolo 6 delle *Note su linguaggio C e dintorni*, ed in particolare il paragrafo 6.2. A mia esperienza, l'accesso ai files è comunemente considerato operazione difficile. Io credo invece che sia solo noiosa, e che la sola reale difficoltà consista nella mancanza di conoscenza su come i files siano strutturati e possano essere utilizzati come deposito di informazioni. D'altra parte, avere dei programmi che producono dati è praticamente inutile se non si sa come memorizzare permanentemente i dati prodotti, e poi riutilizzarli.

I primi due esercizi sono abbastanza semplici, e dovresti essere in grado di svolgerli. Gli altri sono abbastanza complessi, e ti richiederanno un po' di tempo. Ma se riuscirai a svolgerli avrai imparato che con un po' di buona volontà potrai gestire il contenuto di qualunque file.

1. Scrivi una funzione che esegua la copia esatta di un file. Precisamente la funzione dovrà:
- ricevere come argomenti il nome di un file esistente ed il nome di un file nuovo da creare;
- aprire il file esistente in lettura, ed aprire il file nuovo in scrittura;
- copiare byte a byte tutto il file esistente su quello nuovo;
- chiudere ambedue i files;
- restituire come valore intero (meglio se di tipo `long int`) il numero di bytes trasferiti.

Attenzione: Il nome del file in scrittura deve essere diverso da quello in lettura: ricorda che aprire in scrittura un file esistente significa *riscriverlo*, perdendo tutto il contenuto precedente.

L'esercizio sembra complicato, ma se ci pensi attentamente scoprirai che si tratta di una faccenda molto più semplice di quanto appaia a prima vista: dopotutto si tratta solo di leggere un byte dal file di ingresso e scriverlo sul quello di uscita. Qualunque sia il contenuto del file, ti basta considerarlo come binario: non è altro che una sequenza di bit. Il solo problema è come riconoscere la fine del file in ingresso, e fermarsi. Eccoti alcune indicazioni.
- L'apertura dei files viene eseguita da `fopen(`⟨*nome file*⟩`,`⟨*accesso*⟩`)`. Vedi i dettagli nel paragrafo 6.2.4 delle *Note su linguaggio C e dintorni*, oppure la pagina di manuale (`man fopen`). Ricorda che la funzione restituisce un puntatore di tipo `FILE *`.
- La lettura e la scrittura si effettuano con le funzioni `fread` e `fwrite`: vedi il paragrafo 6.2.5. Come ti ho già detto, basta leggere un byte per volta e riscriverlo immediatamente.
- Ricorda che la funzione `fread` restituisce il numero di campi effettivamente letti. Se leggi un byte per volta, continuerà a restituirti `1` fin che ha dati da leggere; quando il file è finito si limita si limita a restituirti `0`.
- La chiusura dei files viene eseguita dalla funzione `fclose`.

Se le indicazioni che ti ho dato non ti bastano prova a guardare il file `copia_file.c` sul dischetto.

La nostra scrittura si fonda sull' uso di un alfabeto, i cui caratteri vengono utilizzati per comporre parole, separate da spazi o da altri segni di interpunzione. Nella rappresentazione al calcolatore i caratteri vengono identificati da numeri, secondo la rappresentazione del codice ASCII. D' altra parte, qualunque file è semplicemente una successione di bytes. Gli esercizi che seguono ti metteranno in grado di fare alcune statistiche semplici sul contenuto di un file, in particolare di un testo. Se non hai a portata di mano un file di testo da analizzare puoi usare `promessi.txt`, che trovi sul dischetto. Come puoi ben immaginare, si tratta del primo capitolo dei *Promessi Sposi* di Alessandro Manzoni.

2. Come primo esercizio prova a scrivere una funzione che calcoli la frequenza con cui compare ciascuno dei 256 caratteri rappresentabili in un byte. Completa l'esercizio con una funzione che stampi il contenuto della tabella su 4 colonne. Naturalmente, dovrai anche scrivere un main che richiami le due funzioni.

Veniamo ai dettagli. La funzione che riempie la tabella si potrà chiamare

```
int contabyte(FILE *in,int cont[]);
```

dove `in` è un puntatore ad un file (che dovrà essere aperto nel momento in cui viene chiamata la funzione) e `cont` è un vettore di 256 interi, destinati a contenere i risultati del conteggio. La funzione dovrà: azzerare la tabella dei contatori, leggere byte a byte l'intero file, e per ciascun byte letto incrementare il contatore corrispondente; nel frattempo dovrà mantenere un conteggio del numero totale di bytes letti, che restituirà alla fine come valore associato al nome. Eccoti le istruzioni rilevanti:

```
unsigned char c;
while(fread(&c,sizeof(char),1,in) != 0)  ++cont[c];
```

Domanda interessante: perché ho dichiarato `unsigned char c`? Il resto della funzione dovresti ormai essere in grado di scriverlo da te.

La funzione di stampa potrà avere come prototipo

```
void stampa_cont(int cont[]);
```

dove `cont` è la tabella di 256 interi riempita da `contabyte`. Qui la difficoltà sta nell'impaginare decentemente la stampa. Sarebbe inteessante scrivere:

(i) il codice ottale del carattere;
(ii) il corrispondente nella codifica `ISO-8859-1` (un' estensione del codice `ASCII` che comprende anche caratteri accentati, &c); naturalmente per molti caratteri questo corrispondente non esiste;
(iii) il numero di volte che quel carattere compare;
(iv) la percentuale di apparizioni rispetto al totale.

Per quanto riguarda il punto (ii) ti sarà utile sapere che i caratteri `ISO-8859-1` che hanno corrispondente stampabile sono quelli con codice decimale compreso negli intervalli `[32-127]` e `[160-255]`. Non affannarti a scoprire come spiegare alla macchina che deve usare il codice `ISO-8859-1`: normalmente è quello che usa come default. Se la faccenda ti creasse troppe difficoltà, risolvi brutalmente il problema stampando solo i caratteri del codice `ASCII`, per intenderci, quelli nell'intervallo `[32-127]`.

Naturalmente, ti troverai con 256 linee di stampa. Sapresti diminuirle stampando 4 risultati per riga? Se vuoi un esempio, compila ed esegui il programma `contabyte.c` che trovi tra i files del dischetto, e prova a riprodurre lo stesso tipo di stampa. Se proprio non ci riesci, prova a curiosare all'interno del file.

Avrai notato che in questo esercizio il fatto che il file contenga un testo è del tutto irrilevante: prova ad eseguire il programma dicendogli di esaminare un file qualunque, ad esempio in file binario (anche un eseguibile, se vuoi). Vedrai che il programma arriverà tranquillamente alla fine e stamperà i risultati, senza preoccuparsi di cosa realmente contenga il file. Ovviamente, le frequenze di apparizione dei caratteri dipendono dal tipo di file che gli passi in input. La ragione è semplice: la funzione `fread()` non esegue alcuna conversione; trasporta il contenuto del file alla memoria senza preoccuparsi del contenuto.

3. Prova ad isolare le parole che compaiono in un file di testo. Un modo comodo consiste nel generare un secondo file che contenga solo le parole del testo, separate da un carattere `NUL`. Alla fine del programma stampa il conteggio del numero di caratteri utili e di parole memorizzate.

Suggerimenti:

- Un file di testo contiene non solo caratteri alfabetici, ma anche segni di interpunzione, spazi di separazione, caratteri `LF` (Line Feed) che separano le righe, parentesi, numeri, caratteri accentati in modo vario (dipende dalla lingua) e quant'altro. La prima operazione da fare è "ripulire" il file da tutti questi caratteri, ed isolare solo le parole. Lo potrai fare con un programma che legge il testo originario e crea un nuovo file col testo pulito. Un modo comodo è usare una tabella di conversione che fa corrispondere a ciascun byte un codice di codifica "ad uso personale". Va da sè che costruire la tabella è l'operazione più lunga e noiosa — anche perché richiede la conoscenza del famigerato codice `ISO-8859-1`. Non preoccuparti: l'ho già fatto io: cerca la tabella nel file `parole.c`; avrai anche modo di vedere come si possano inizializzare dei vettori allocati in area globale. Ecco i criteri usati nel costruire la tabella:
    - per i caratteri maiuscoli del codice `ASCII` viene mantenuto il codice originario;
    - i caratteri minuscoli vengono trasformati nel corrispondente maiuscolo;
    - i caratteri accentati o contrassegnati in vario modo (ad esempio la c con cediglia) vengono trasformati nel corrispondente carattere maiuscolo privato dell' accento (o contrassegno che dir si voglia);
    - a tutti gli altri caratteri viene fatto corrispondere il codice `NUL` (definito come zero binario);

Naturalmente i criteri seguiti sono in larga misura arbitrari. Ad esempio, è perfettamente lecito considerare i caratteri accentati come effettivamente diversi da quelli non accentati, o adottare una codifica personale dei caratteri alfabetici che si ritenga più conveniente ai propri scopi. Ma in tal caso tutto si riduce a modificare la tabella.

Il file `parole.c` contiene anche il programmino di conversione. Il file risultante contiene solo delle parole separate da un singolo carattere `NUL`. In altre parole, le sequenze di più caratteri `NUL` consecutivi generati dalla conversione tramite tabella vengono ridotte ad un solo `NUL` che fa da separatore tra le parole.

Questa volta potrai ben provare a scrivere tu stesso il programma, ma credo che una sbirciatina al file non ti farà male: qualche piccolo trucco del mestiere ci vuole.

4. Utilizzando il file prodotto nell' esercizio precedente, scrivi un programma che esegua le operazioni seguenti:

i. crea in memoria una tabella che contenga tutto il file; a tal fine ti basta allocare una tabella di grosse dimensioni (ad esempio, 1 Megabyte) e limitarti a controllare che le dimensioni del file non superino quelle della tabella;

ii. ripeti la statistica sulla frequenza dei caratteri del primo esercizio, ma questa volta isolando i soli caratteri alfabetici (i `NUL` che separano le parole non vanno conteggiati);

iii. fa una statistica sulla lunghezza delle parole: dovrai stampare, ad esempio, il numero di parole che hanno rispettivamente lunghezza 1,2,. . .,20, e quante parole sono più lunghe di 20 caratteri (ad esempio, *precipitevolissimevolmente* o *supercalifragilistichespiralidoso*);

iv. analizza la correlazione tra i caratteri creando una matrice che dica quante volte ad un determinato carattere ne segue un altro; se tieni conto che il file contiene solo caratteri alfabetici maiuscoli più il `NUL` di separazione delle parole ti servirà una matrice 27×27.

Le quattro operazioni richieste dovranno essere codificate in altrettanti sottoprogrammi, richiamati dal programma principale.

Suggerimenti:

- Questo programma è decisamente più complesso di quello che ti sono stati richiesti fin qui. Sfrutta la suddivisione in sottoprogrammi: non è necessario che il programma sia completo per poterlo eseguire. Puoi procedere in più fasi.
  a. Scrivi il `main` inserendo le chiamate ai sottoprogrammi, e completa il programma inserendo dei sottoprogrammi vuoti; durante l' esecuzione non accadrà nulla, ma almeno saprai che la struttura generale del programma è corretta.
  b. Scrivi i sottoprogrammi uno per volta, e provali separatamente.
- Per l' esercizio iv (la correlazione) dovrai costruire una matrice di contatori, diciamo `corr[j][k]`. Ogni riga `corr[j]` corrisponde ad un carattere (non dimenticare il `NUL` che separa le parole, e che potrai, volendo, interpretare come uno spazio). Ogni elemento `corr[j][k]` della riga conta quante volte il carattere `j`–esimo è seguito dal carattere `k`–esimo. Poi dovrai mantenere un totale dei conteggi per ciascuna riga, ed alla fine stampare quante volte in percentuale il carattere `j`–esimo è seguito dal carattere `k`–esimo. La tabella da stampare risulta eccessivamente larga, ma ti basterà stampare un certo numero di colonne per volta (ad esempio 9). Nello stampare la tabella non dimenticare che anche il `NUL` (o lo spazio) di separazione è un carattere. Ad esempio, se `A` è seguito da `NUL` significa che una parola finisce per `A`; se `NUL` precede `A` significa che una parola inizia per `A`.
- Se dopo una settimana non sei riuscito a svolgere l'esercizio, prova a guardare il file `statcar.c` sul dischetto.