ESERCITAZIONE 11 APRILE 2024

NOTA BENE: per ognuno dei programmi C che devono essere scritti per questa esercitazione (e per le prossime) deve essere scritto anche il makefile che produce il file eseguibile controllando tutti i possibili WARNING di compilazione (opzione –Wall) che devono sempre essere eliminati, come chiaramente anche gli errori di compilazione e di linking! Inoltre, si faccia attenzione al fatto che se si usa un particolare standard del linguaggio, anche questo va specificato dato che non si può presupporre che chi deve verificare il funzionamento abbia lo stesso standard di default!

1) Con un editor, scrivere un programma in C `parametri1.c` che deve essere invocato con *almeno 1 parametro*; dopo aver fatto il controllo che il numero dei parametri sia giusto (controllo LASCO), si visualizzi su standard output il nome dell'eseguibile e il numero dei parametri e, quindi, il valore di ogni parametro (cioè la corrispondente stringa) inserendo, contestualmente, il numero d'ordine dei singoli parametri: attenzione che il primo vero parametro è argv[1] e quindi il suo indice è **1**; se si indica con **N** il numero di parametri, si imposti il ciclo for con un indice **i** che varia da **0** a **N-1** (quindi con **N** escluso) e quindi i parametri si otterranno utilizzando su argv l'indice corrispondente a **i+1** (**IL PERCHÈ DI QUESTA SCELTA SARÀ CHIARO IN SEGUITO**). Si verifichi il funzionamento sia in assenza di parametri (caso errato) che con un numero variabile di parametri.
2) Con un editor, scrivere un programma in C `parametri2.c` che deve essere invocato *esattamente con 3 parametri*: il *primo* parametro deve essere considerato il nome di un file, il *secondo* un numero intero **N** strettamente positivo e il *terzo* deve essere considerato un singolo carattere **C**. Dopo aver fatto il controllo che il numero dei parametri sia giusto (controllo STRETTO), si devono controllare anche i 'tipi' dei parametri: per il controllo che il *primo* parametro sia il nome di un file si può usare la primitiva *open* in lettura e in caso di insuccesso uscire (come visto negli esercizi mostrati a lezione); per il controllo che il *secondo* parametro sia un numero intero si può usare la funzione di libreria *atoi* e sul valore risultante **N** verificare che se minore o uguale a 0 uscire; per il controllo che il *terzo* parametro sia un singolo carattere si può scegliere *a)* se usare la funzione di libreria *strlen* (in questo caso ricordarsi di includere l'header file con le funzioni per operare sulle stringhe) e sul valore risultante verificare che se diverso da 1 uscire *oppure b)* si può verificare che il secondo carattere della stringa (argv[3][1]) se diverso da '\0' (*terminatore di stringa*) uscire. Dopo aver fatto tutti i controlli, si visualizzi su standard output il nome dell'eseguibile e il numero dei parametri (come nell'esercizio precedente) e, quindi, il valore dei tre parametri (**secondo il loro significato**) inserendo opportune frasi che facciano capire all'utente che cosa si sta visualizzando. Si verifichi il funzionamento sia in assenza di parametri o con un numero minore di 3 o maggiore di 3 (casi errati) che con il numero giusto di parametri, ma di 'tipo' non corretto (file non esistente, stringa che non corrispondente ad un numero > 0, stringa NON costituita da un solo carattere) e infine il funzionamento corretto.
3) Scaricare da **GITHUB** uno qualunque degli esercizi visti a lezione (ad esempio il file `copia.c`) e, usando un editor, andare a commentare *una alla volta* le direttive di *include*; provare ad utilizzare l'*utility make* per andare a verificare quanto scritto nella **slide 14 C/UNIX Primitive per i file**.
4) Con un editor, scrivere un programma in C `provaBUFSIZ.c` che semplicemente visualizzi su standard output il valore della costante **BUFSIZ** usata, ad esempio, nel file visto a lezione `copia.c`.
5) Con un editor, scrivere un programma in C `contaOccorrenze.c` che deve essere invocato esattamente con 2 parametri: il *primo* deve essere considerato il nome di un file (**F**), mentre il secondo deve essere considerato un singolo carattere **Cx**. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri e sul loro 'tipo'), si visualizzi su standard output il nome dell'eseguibile e i parametri e quindi si calcoli quante occorrenze (*in termini di long int*) del carattere **Cx** sono presenti nel file **F** e si visualizzi tale valore su standard output inserendo opportune frasi che facciano capire all'utente che cosa si sta visualizzando.
   N.B. La lettura dal file, in questo caso, DEVE essere fatta *un* carattere alla volta dato che si deve individuare uno specifico carattere, come spiegato a lezione. Si verifichi il funzionamento sia con il numero non corretto di parametri che con il numero giusto di parametri, ma di 'tipo' non corretto e infine il funzionamento corretto.
6) Con un editor, scrivere un programma in C `mycat1.c` che, partendo dal programma `mycat.c` mostrato a lezione (disponibile su **GITHUB**), che deve funzionare con un numero qualunque di nomi di file passati come parametri (anche nessuno, come previsto anche nel programma `mycat.c`!) e che quindi si comporti in tutto e per tutto come il comando/filtro *cat*.
   N.B. Se il numero di parametri è uno o più di uno, risulta necessario impostare un ciclo all'interno del quale si deve prevedere di eseguire la *open* del file corrente e quindi la visualizzazione del contenuto del file (quest'ultima parte esattamente uguale a quella del programma `mycat.c`); NON importa, quindi, prevedere un array dinamico di file descriptor! Si verifichi il funzionamento.

7) Con un editor, scrivere un programma in C `selezionaMultipli.c` che deve essere invocato *esattamente con 2 parametri*: il primo deve essere considerato il nome di un file **F**, mentre il secondo un numero intero **n** strettamente positivo. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri e sul loro 'tipo'), si visualizzi su standard output tutti i caratteri del file **F** che si trovano in posizione multipla di **n** insieme con l'indicazione del numero di tale multiplo (ad esempio scrivendo su standard output "Il carattere multiplo *x*-esimo all'interno del file XXX e' *y*" dove *x* è il numero del multiplo, XXX il nome del file e *y* è il carattere). Si verifichi il funzionamento.
   OSSERVAZIONE: Questo esercizio può essere risolto in due modi:
   a) La lettura può avvenire a multipli di **n** (quindi via via una lettura di **n** caratteri in una volta sola!) e tutte le volte che si riescono a leggere **n** caratteri si riporta sullo standard output appunto l'**n**-esimo carattere. **N.B.** In questo caso bisogna allocare *dinamicamente* un array di **n** caratteri (NON SI POSSONO USARE I VARIABLE LENGHT ARRAY!) e controllare l'avvenuta creazione! ➔ questa versione la si chiami `selezionaMultipli1.c`
   b) Dopo aver calcolato la lunghezza in caratteri del file utilizzando la primitiva *lseek*, si può impostare un ciclo all'interno del quale si va a calcolare la posizione multipla di **n** a cui spostarsi e, se questa è corretta, sempre con la primitiva *lseek* si deve spostare il File Pointer nella posizione giusta e quindi si può leggere direttamente il singolo carattere cercato! ➔ questa versione la si chiami `selezionaMultipli2.c`
   Si verifichi che i caratteri scritti dalle due versioni siano gli stessi!
8) Con un editor, scrivere un programma in C `sostituisciCar.c` che deve essere invocato *esattamente con 2 parametri*: il primo deve essere considerato il nome di un file **F**, mentre il secondo deve essere considerato un singolo carattere **Cx**. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri e sul loro 'tipo'), si deve operare una modifica del contenuto del file **F**: in particolare, tutte le occorrenze del carattere **Cx** nel file **F** devono essere sostituite con il carattere spazio/blank.
   N.B. Si faccia attenzione che, quando si trova una occorrenza del carattere **Cx** (andando a leggere dal file un carattere alla volta, come nel programma `contaOccorrenze.c`), il File Pointer risulta già avanzato e quindi bisogna portarlo indietro di 1 (con la primitiva *lseek*) prima di poter fare la scrittura del carattere spazio!
9) Con un editor, scrivere un programma in C `sostituisciCar1.c` che, partendo dal programma `sostituisciCar.c` precedente, consideri un ulteriore parametro (**Change**) che deve essere ancora un singolo carattere (come il secondo parametro) che deve essere usato per sostituire il carattere cercato: adattare chiaramente i vari controlli a questa situazione. Si verifichi il funzionamento, anche nel caso che il carattere passato come terzo parametro sia il carattere spazio/blank!
10) Scrivere un programma in C `myhead1.c` che deve essere invocato *esattamente con un parametro* che deve essere considerato come la tipica opzione del **comando-filtro** *head* e quindi **-n** *(dove n deve essere un numero)*. Nei controlli da effettuare si deve considerare di controllare che il primo carattere dell'unico parametro passato deve essere il carattere '**-**'; quindi la conversione da stringa a numero (con relativo controllo che tale numero sia strettamente positivo e valido, come fatto nei precedenti esercizi) deve prendere in considerazione la stringa il cui indirizzo iniziale è `&(argv[1][1])`. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri e sul loro 'tipo'), il programma si deve comportare come il **comando-filtro** *head* e quindi deve filtrare in uscita le prime *numero* linee dello standard input.
    N.B. La lettura deve avvenire, anche in questo caso, un carattere alla volta, dato che si deve cercare il carattere '**\n**' riportando sempre contestualmente il carattere letto (*anche lo '\n'*) sullo standard output; il contatore del numero di linee lo si faccia partire da **1** e quindi quando i caratteri '\n' incontrati saranno arrivati a superare **n** si potrà concludere il ciclo di lettura, ad esempio con un *break*. Si verifichi il funzionamento.
11) Scrivere un programma in C `myhead2.c` che, partendo dal programma `myhead1.c`**,** *può essere invocato con una opzione* ***-n*** *(come il precedente) o anche senza alcun parametro*: in questo secondo caso, deve essere considerato di filtrare le prime 10 linee, esattamente come si comporta il **comando-filtro** *head*. Si verifichi il funzionamento.
12) Scrivere un programma in C `myhead3.c` che, partendo dal programma `myhead2.c`**,** si comporti sia come comando-filtro che come comando (filtrando le linee del file specificato invece che quelle dello standard input) e quindi possa avere *fino a due parametri* (il primo deve essere considerato l'opzione **-n** e il secondo il nome del file). Si verifichi il funzionamento.

13) Scrivere un programma in C `selezionaLinea.c` che deve essere invocato *esattamente con 2 parametri*: il primo deve essere considerato il nome di un file **F**, mentre il secondo un numero intero **n** strettamente positivo. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri e sul loro 'tipo'), si visualizzi su standard output la linea n-esima del file **F** (*si può supporre la lunghezza massima di ogni linea del file pari a 255 caratteri*) se esiste, *oppure* si riporti che tale linea non esiste.
N.B. La lettura deve avvenire, anche in questo caso, un carattere alla volta, dato che si deve cercare il carattere '**\n**', ma in questo caso i caratteri devono essere salvati in un buffer di opportuna lunghezza (256 perché si considera utile inserire anche il terminatore di stringa quando si sarà identificata la linea giusta); il modo più furbo per farlo è questo: `while (read (fd, &(buffer[j]), 1) != 0)` supponendo che `j` sia l'indice, inizializzato a 0, che scorre gli elementi del buffer. Inoltre, anche in questo caso, il contatore del numero di linee lo si faccia partire da **1** e quindi quando i caratteri '\n' incontrati saranno arrivati a **n** si potrà concludere il ciclo di lettura, ad esempio con un *break*, dopo aver chiaramente stampato la linea cercata (N.B. per usare la printf, bisogna trasformare la linea in una stringa!) e aver cambiato valore alla variabile `trovata` (si veda dopo); se invece non viene trovata la linea, basandosi appunto sul valore della variabile `trovata`, si deve dare una indicazione all'utente in tal senso! Si verifichi il funzionamento.

14) Scrivere un programma in C `selezionaLunghezzaLinea.c` che deve essere invocato *con esattamente 2 parametri*: il primo deve essere considerato il nome di un file **F**, mentre il secondo un numero intero **n** strettamente positivo. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri e sul loro 'tipo'), si visualizzino su standard output tutte le linee del file **F** la cui lunghezza, *compreso il terminatore di linea*, sia esattamente uguale a **n** (insieme con la indicazione del numero d'ordine di tale linea) oppure si riporti che non esiste alcuna linea con lunghezza uguale a **n**.
N.B. La soluzione è molto simile al programma `selezionaLinea.c`, ma in questo caso non si deve prevedere un *break* nel ciclo di lettura, dato che potremmo avere più linee con la stessa lunghezza richiesta; si faccia attenzione che se si deve contare anche il terminatore di linea, allora una volta individuato il carattere '\n' in `buffer[j]` il controllo sulla lunghezza della linea dovrà essere `n == j+1`. Si verifichi il funzionamento.