Esercitazione di laboratorio n. 1

Esercizio n.1: Valutazione di espressioni regolari

I problemi di ricerca di stringe all'interno di testi e/o collezioni di stringhe (solitamente di dimensione maggiore rispetto alla stringa cercata) si basano raramente su un confronto esatto, ma molto spesso necessitano di rappresentare in modo compatto non una, ma un insieme di stringhe cercate, evitandone per quanto possibile l'enumerazione esplicita. Le ***espressioni regolari*** sono una notazione molto utilizzata per rappresentare (in modo compatto) insiemi di stringhe correlate tra loro (ad esempio aventi una parte comune).
Una ***espressione regolare*** (o regexp) è una sequenza di simboli (quindi una stringa) che identifica un insieme di stringhe. Si scriva una funzione in C in grado di individuare (cercare) eventuali occorrenze di una data regexp all'interno di una stringa di input.
La funzione sia caratterizzata dal seguente prototipo:

```
char *cercaRegexp(char *src, char *regexp);
```

dove :

- il parametro `src` rappresenta la stringa sorgente in cui cercare.
- il parametro `regexp` rappresenta l'espressione regolare da cercare.
- il valore di ritorno della funzione è un puntatore alla prima occorrenza di `regexp` in `src` (`NULL` se non trovata).

Ai fini dell'esercizio si consideri di valutare solamente stringhe composte da caratteri alfabetici. Si considerino inoltre solamente espressioni regolari composte da caratteri alfabetici e dai seguenti metacaratteri:

- . trova un singolo carattere (cioè qualunque carattere può corrispondere a un punto)
- [] trova un singolo carattere contenuto nelle parentesi (cioè uno qualsiasi dei caratteri tra parentesi va bene)
- [^] trova ogni singolo carattere non contenuto nelle parentesi (cioè tutti i caratteri tra parentesi non vanno bene)
- \a trova un carattere minuscolo
- \A trova un carattere maiuscolo

Esempi di espressioni regolari:
`.oto` corrisponde a ogni stringa di quattro caratteri terminante con "`oto`", es. "`voto`", "`noto`", "`foto`", ...
`[mn]oto` rappresenta solamente "`moto`" e "`noto`"
`[^f]oto` rappresenta tutte le stringhe terminanti in "`oto`" ad eccezione di "`foto`"
`\aoto` rappresenta ogni stringa di quattro caratteri (come "`voto`", "`noto`", "`foto`", ...) iniziante per lettere minuscola e terminante in "`oto`"
`\Aoto` rappresenta ogni stringa di quattro caratteri (come "`Voto`", "`Noto`", "`Foto`", ...) iniziante per lettere maiuscola e terminante in "`oto`".

NOTA: i metacaratteri possono apparire in qualsiasi punto dell'espressione regolare. I casi qui sopra sono solo una minima parte a titolo di esempio. Sono quindi espressioni regolari valide: `A[^f]\anR.d`, `\A[aeiou]5t[123]`, ecc.

Esercizio n.2: Azienda di trasporti
Competenze: selezione/filtro dati mediante ricerca in tabelle di nomi/stringhe, tipi enumerativi
Questo esercizio è stato già proposto nel Lab. 6 di Tecniche di Programmazione dell'a.a. 2020/21.

Un'azienda di trasporti urbani traccia i propri automezzi in un file di log (file testuale di nome corse.txt). Il file è organizzato come segue:

- sulla prima riga, un intero positivo che indica il numero di successive righe del file stesso (al più 1000)
- nelle righe successive le informazioni sulle tratte, uno per riga, con formato:
  `<codice_tratta><partenza><destinazione><data><ora_partenza><ora_arrivo><ritardo>`

Tutte le stringhe sono lunghe al massimo 30 caratteri. Il ritardo è un numero intero, eventualmente nullo, a rappresentare i minuti di ritardo accumulati dalla corsa.
Si scriva un programma C in grado di rispondere alle seguenti interrogazioni:

1. elencare tutte le corse partite in un certo intervallo di date
2. elencare tutti le corse partite da una certa fermata
3. elencare tutti le corse che fanno capolinea in una certa fermata
4. elencare tutte le corse che hanno raggiunto la destinazione in ritardo in un certo intervallo di date
5. elencare il ritardo complessivo accumulato dalle corse identificate da un certo codice di tratta

Le interrogazioni di cui sopra siano gestite mediante menu di comandi (si veda il paragrafo 4.4.1, Dal problema al programma). Ogni comando consiste di una parola tra "date", "partenza", "capolinea", "ritardo", "ritardo_tot" e "fine", eventualmente seguita sulla stessa riga da altre informazioni, ad esempio due date per "date", una fermata di partenza per "partenza", etc.
Si utilizzi la strategia di codifica dei comandi mediante tipo `enum comando_e`, contenente i simboli `r_date`, `r_partenza`, `r_capolinea`, `r_ritardo`, `r_ritardo_tot`, `r_fine`, che consente menu basati su `switch-case`.
Si consiglia di:

- realizzare una funzione `leggiComando` che, acquisito in modo opportuno il comando, ritorni il corrispondente valore di tipo `comando_e`
- realizzare una funzione `selezionaDati` che, ricevuti tra i parametri la tabella, la dimensione della tabella e il tipo di comando, gestisca mediante menu l'acquisizione delle informazioni aggiuntive necessarie per quel comando e la chiamata di un'opportuna funzione di selezione e stampa dei dati selezionati.

Esercizio n.3: Azienda di trasporti - ordinamento

Si consideri lo scenario introdotto nell'esercizio precedente. Si realizzi un programma in C che, una volta acquisite le informazioni in una opportuna struttura dati, renda disponibili le seguenti operazioni:

- stampa, a scelta se a video o su file, dei contenuti del log
- ordinamento del vettore per data, e a parità di date per ora
- ordinamento del vettore per codice di tratta
- ordinamento del vettore per stazione di partenza
- ordinamento del vettore per stazione di arrivo
- ricerca di una tratta per stazione di partenza (anche parziale come prefisso della stringa, quindi dal primo carattere). Nel caso di più risultati, elencarli tutti.

Per quanto riguarda le ricerche, si richiede che siano implementate sia una funzione di ricerca dicotomica sia una funzione di ricerca lineare. Per quanto riguarda l'ordinamento, si presti attenzione alla stabilità dell'algoritmo prescelto nel caso di ordinamento per più chiavi successive.

Esercizio n.4: Azienda di trasporti - multiordinamento

A partire dalle specifiche dell'esercizio precedente, estendere le funzionalità del programma per mantenere in contemporanea più ordinamenti della base dati
Suggerimento: si legga il vettore originale una sola volta, mantenendolo nell'esatto ordine di lettura per tutta la durata dell'esecuzione. Si affianchino al vettore originale tanti vettori di puntatori a struttura quanti sono gli ordinamenti richiesti, con i quali sono gestiti gli ordinamenti.