Esercitazione di laboratorio n. 4

(Caricamento sul portale entro le 23.59 del 02/12/2017 dell'esercizio 2 e di 2 tra gli es. 1, 3 e 4)

Esercizio n. 1: Individuazione di regioni (rivista)

Competenze: lettura/scrittura di file, manipolazioni di matrici statiche; puntatori e passaggio di parametri per riferimento (Puntatori e strutture dati dinamiche: 1.4)
Categoria: problemi di verifica e selezione (Dal problema al programma: 4.5)

Si riveda l'esercizio 1 del laboratorio 3 apportandovi le seguenti modifiche:

- supponendo di avere dichiarato una matrice di interi `M` e di aver definito `MAXR` come `50` si acquisisca la matrice mediante una funzione (`leggiMatrice`) che ne ritorna il numero di righe e di colonne effettivamente usati, come parametri "by reference" (o meglio, con puntatori by value). La funzione deve poter essere chiamata con un'istruzione del tipo:
```
leggiMatrice(M,MAXR,&nr,&nc);
```
- per effettuare il riconoscimento delle regioni si utilizzi una funzione `riconosciRegione` che, data una casella della matrice, determini se si tratti o meno di estremo superiore sinistro di una regione, ritornandone "by reference" (come per la precedente) le dimensioni del rettangolo, e avente come valore di ritorno un intero booleano (vero: rettangolo trovato, falso: rettangolo non trovato). La funzione deve poter essere chiamata come segue:
```
if (riconosciRegione(M,nr,nc,r,c,&b,&h)) {
  // stampa messaggio per rettangolo con
  // estremo in (r,c), base b e altezza h
  ...
}
```

Esercizio n. 2: Occorrenze e manipolazione di parole

Competenze: lettura/scrittura di file, manipolazioni di testi, ricerca in tabelle di nomi/stringhe, puntatori a carattere, dualità puntatore vettore, ordinamento di vettore di puntatori.
Categoria: problemi di elaborazione testi mediante stringhe (Dal problema al programma: 4.4.3), puntatori e sott-vettori, vettori e matrici come parametri, puntatori a stringhe, vettori di puntatori (Puntatori e strutture dati dinamiche: 2.1, 2.2, 2.3, 2.4)

Si scriva un programma in grado di localizzare, all'interno di un generico testo, le occorrenze di ogni parola, definita come una sequenza massimale di caratteri alfabetici consecutivi: una parola inizia quindi con un carattere alfabetico preceduto (se c'è) da un carattere non alfabetico e seguita da un carattere non alfabetico (o dalla fine del testo). Sulle parole presenti nel testo sia quindi possibile effettuare ricerche di tipo logaritmico, mediante una funzione di ricerca binaria.

Più in dettaglio:

- il testo viene acquisito da un file `sequenze.txt`, contenente una pagina di non più di 60 righe di al più 80 caratteri (a-capo incluso, a tale scopo si definiscano le costanti MAXR e MAXC). Il file va letto una sola volta, caricandone il contenuto in una matrice di caratteri di dimensioni opportune, corrispondente alla definizione:
```
char pagina[MAXR][MAXC+1];
```

mediante una funzione `leggiPagina`, richiamabile come segue (il valore ritornato è il numero effettivo di righe lette):

```
nr = leggiPagina(pagina,MAXR);
```

- una volta acquisito il testo, il programma deve scandire tutte le parole presenti (si sa che non sono più di 1000 e a tale scopo si definisca la costante `MAXP`), accumulando i puntatori a tali parole (per ognuna il puntatore al carattere iniziale della parola in `pagina`), nel vettore
  `char *parole[MAXP];`
  mediante una chiamata a funzione del tipo (il valore ritornato è il numero effettivo di parole riconosciute):
  `np = riconosciParole(pagina,nr,parole,MAXP);`
  Dopo tale chiamata, le prime `np` caselle del vettore `parole` contengono i puntatori alle parole (o meglio al primo carattere di ogni parola) in `pagina`

- si effettui sul vettore `parole` un ordinamento (basato su confronto che ignori la differenza tra maiuscole e minuscole) di tipo INSERTION SORT. Si noti che occorre riordinare dei puntatori e che il confronto tra due parole va fatto con una funzione specifica `confrontaParole`, che deve essere appositamente realizzata, in quanto le parole in `pagina` non terminano con `'\0'`. L'ordinamento deve essere stabile, in quanto, nel caso di parole uguali, i loro puntatori devono garantire l'ordinamento iniziale nel testo

- successivamente, il programma proponga all'utente una ricerca ripetuta di parole all'interno del testo, che termini una volta immessa la parola `"$fine"`. Per ogni parola immessa (variabile `cerca`, contenente una stringa), la si deve cercare, utilizzando una funzione di ricerca binaria (dicotomica), richiamabile come:
  `indice = ricercaBinaria(parole,np,cerca);`
  la funzione ritorna l'indice in `parole`, eventualmente -1 (per indicare parola non trovata) corrispondente alla posizione della parola cercata. Nel caso di occorrenze multiple, l'indice deve localizzare una qualunque delle occorrenze, a partire dalla quale il programma chiamante deve successivamente localizzare (mediante un'iterazione all'indietro e una in avanti), la prima e l'ultima delle occorrenze della parola cercata.
  Per ognuna delle occorrenze della parola cercata va stampato l'indice di riga e di colonna di inizio della parola, nella matrice `pagina`: se ad esempio la parola Italia comparisse 3 volte, in riga e colonna (2,5), (5,0) e (10,14), occorrerebbe stampare (si utilizzino indici a partire da 0)
  `Parola Italia trovata in (2,5), (5,0), (10,14).`

Suggerimento: per il calcolo degli indici di riga e colonna di inizio di una parola, si utilizzi l'aritmetica dei puntatori, ricordando che una matrice è organizzata secondo la strategia ***row-major***, e che un carattere alla riga `r` colonna `c` (gli indici iniziano da 0), si trova in `pagina` dopo le prime `r` righe (complete) e `c` caratteri della riga `r`.

Esercizio n. 3: Sequenza H di Hofstadter

La sequenza H di Hofstadter è una sequenza ricorsiva definita come:

$$H(0) = 0$$

$$H(n) = n - H\Big(H\big(H(n-1)\big)\Big), n > 0$$

I primi termini di tale sequenza sono:

```
0, 1, 1, 2, 3, 4, 4, 5, 5, 6, 7, 7, 8, 9, 10, 10, 11, 12, 13, 13, 14, 14, 15,
16, 17, 17, 18, 18, 19, 20, 20, 21, 22, 23, 23, 24, 24, 25, 26, 26, 27, 28, 29,
29, 30, 31, 32, 32, 33, 33, 34, 35, 35, 36, 37, 38, 38, 39, 40, 41, 41, 42, 42,
43, 44, 45, 45, 46, 46, 47, 48, 48, 49, 50, …
```

Si scriva un programma in C che, ricevuto un intero positivo `N`, stampi i primi `N` termini della sequenza stessa.

Esercizio n. 4: Somme ricorsive

Dato un numero intero non negativo `N`, si scriva una funzione ricorsiva in C che permetta di restituire la somma delle cifre che compongono il numero stesso.
Per la risoluzione di questo esercizio non è ammesso l'uso di costrutti iterativi.

Esempio:
`somma(1289);` deve ritornare 20 (1+2+8+9)