Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Dati aggregati - esercizi da svolgere in laboratorio

1 Cifre ripetute di un numero

Scrivete un programma che legga in input un numero intero n usando `scanf( "%d", &n )` e stabilisca se n contiene cifre ripetute e in caso affermativo quali.

Suggerimento: ricordate la struttura dati usata per prendere gli ordini della pizza!

2 Da base b a base 10

Scrivere un programma che dato un numero b (in base 10) e una sequenza s di cifre in $\{0, \ldots b-1\}$ terminata da un punto, stampi il numero la cui rappresentazione in base b è data da s. Potete assumere che il numero di cifre di s sia sempre minore di 100.

Esempio di funzionamento:

```
Inserisci un intero b e un numero in base b da convertire in base 10: 3 211.
Il numero 211 in base 3 equivale al numero 22 in base 10.
```

NOTA: *è proprio necessario usare un array per svolgere questo esercizio?*

3 Istogramma

Scrivete un programma che legga una sequenza di caratteri terminata da un punto e che visualizzi un istogramma con una barra per ogni carattere dell'alfabeto, lunga quanto il numero delle sue occorrenze. Il programma non deve visualizzare le barre delle lettere che non compaiono e non deve fare distinzione fra maiuscole e minuscole (a tal fine potete usare le funzioni dichiarate nel file `ctype.h`).

Suggerimento: la struttura dati usata per prendere gli ordini della pizza non vi fa venire in mente niente?

Esempio di funzionamento:

```
C'era un RAGAZZO che come ME amava i Beatles e I rolling StoneS.
A *******
B *
C ***
E ********
G **
H *
I ***
L ***
M ***
N ***
O ****
R ***
S ***
T **
U *
V *
Z **
```

4 Anagrammi

Due parole costituiscono un *anagramma* se l'una si ottiene dall'altra permutando le lettere (es: attore, teatro). Scrivete un programma che legga due parole e verifichi se sono anagrammi.

Suggerimento: sfruttate il programma scritto per l'esercizio precedente!

5 Schiacciatina

Scrivete un programma che legga due interi *r*, *c*, seguiti da una matrice di *r* righe e *c* colonne contenente lettere maiuscole e asterischi, e che stampi in output la matrice che si ottiene da quella in input *schiacciando verso il basso* le lettere e *facendo galleggiare* gli asterischi. Ad esempio, se la matrice è data da

```
V * S
* * B
K * *
* S *
```

il programma dovrà stampare la matrice seguente:

```
* * *
* * *
V * S
K S B
```

Potete assumere che *r* e *c* siano al massimo pari a 100.

6 Rubrica

Scrivete un programma per la gestione di una semplice rubrica: attraverso un menu l'utente deve poter visualizzare la rubrica e inserire nuovi numeri. Ogni voce della rubrica deve essere composta almeno da un nominativo e da un numero di telefono; documentate con opportuni commenti eventuali assunzioni sulla lunghezza massima delle stringhe e sul numero massimo di voci presenti in rubrica.

7 Ordinamento per inserimento

Scrivete una programma che legga da standard input una sequenza di interi distinti terminati da 0 (potete assumere che la sequenza sia formata al più 100 numeri), memorizzandoli in un vettore ordinato: ogni volta che viene letto un nuovo intero, il vettore viene scorso fino a trovare l'esatta collocazione del numero, quindi si crea lo spazio per il nuovo numero spostando in avanti i numeri successivi già memorizzati.

Adattate il programma precedente e integratelo con il programma per la gestione della rubrica, in modo che l'inserimento delle nuove voci avvenga in ordine alfabetico (rispetto al nominativo). Per confrontare due stringhe rispetto all'ordine alfabetico, potete usare la funzione `strcmp` della libreria `string.h`.

8 Passeggiate aleatorie

Scrivete un programma che generi una "passeggiata aleatoria" (in inglese *random walk* in un array bidimensionale di dimensione 10×10. L'array sarà riempito di caratteri (inizialmente da punti). Il programma dovrà muoversi di elemento in elemento spostandosi ogni volta di un passo in direzione su, giù, destra o sinistra. Gli elementi visitati andranno etichettati con le lettere dalla A alla Z, nell'ordine in cui vengono visitati. E' importante controllare ad ogni passo che la passeggiata non esca dall'array e che non ritorni su posizioni già visitate. Quando si verifica una di queste condizioni, provate in altre direzioni. Se tutte e quattro le direzioni sono bloccate, il programma deve uscire.

Esempio di funzionamento completo

```
A . . . . . . . . .
B C D . . . . . . .
. . E . . . . . . .
. . F . . . . . . .
I H G . . . . . . .
J . . . . S T . . .
K N O P Q R U . . .
L M . . Z Y V . . .
. . . . . X W . . .
. . . . . . . . . .
```

Esempio di uscita prematura dal programma

```
A . . . . . . . . .
B C D . . . . . . .
K L E . . . . . . .
J M F . . . . . . .
I H G . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
```

Suggerimenti

- Per generare a caso una direzione potete usare le funzioni`time` (da `time.h`), `rand` e `srand` (da `stdlib.h`). La chiamata funzione `rand()` produce un numero apparentemente casuale, ma generato in realtà a partire da un seme. La funzione `srand(n)` inizializza il seme; se il seme non viene inizializzato, il suo valore di default è 1. La chiamata della funzione `time(NULL)` restituisce data e ora corrente, codificate come un unico intero; con la chiamata `srand(time(NULL))` è possibile differenziare i semi e quindi garantire che la passeggiata sia diversa ad ogni esecuzione del programma.

- Dopo aver generato un numero con `rand()`, considerate il suo resto modulo 4. I quattro possibili valori (0,1,2,3) possono essere usati per indicare le direzioni (rappresentatele con una variabile di tipo enum!).