Laboratorio di Algoritmi e Strutture Dati

Docente: Camillo Fiorentini

4 dicembre 2007

Esercizio 1

L'obiettivo è di scrivere un programma `register.c` per gestire un registro di prenotazione di posti numerati da 0 a $n-1$. Il valore di n (numero dei posti prenotabili) è inserito dall'utente all'atto della creazione del registro.

Un cliente è identificato da una stringa di lunghezza massima `WORD`, dove il valore di `WORD` è definito nel programma.

Deve essere possibile compiere le seguenti operazioni.

- **newRegister** (n)

  Crea un nuovo registro che permetta la prenotazione di n posti, da 0 a $n-1$.

  Se esiste già un registro di prenotazione, quest'ultimo deve essere cancellato.

- **book**$(k, name)$

  Se il posto k è libero, prenota il posto k per il cliente identificato da *name*. Altrimenti, stampa un messaggio di errore.

- **cancel**(k)

  Se il posto k è occupato, cancella la prenotazione di k. Altrimenti, stampa un messaggio di errore.

- **move**$(from, to)$

  Sposta il cliente dal posto $from$ al posto *to* se ciò è possibile (ossia, $from$ è occupato e *to* libero). Altrimenti, stampa un messaggio di errore.

- **printRegister**()

  Stampa il contenuto del registro (posti prenotati).

Si noti che l'implementazione in C delle precedenti operazioni di alto livello può richiedere l'uso di parametri in più rispetto a quelli indicati.

Il programma deve leggere da standard input una sequenza di istruzioni secondo il formato nella tabella, dove k, n, $from$ e *to* sono interi e *name* una parola di lunghezza massima `WORD`.

I vari elementi sulla riga sono separati da uno o più spazi. Quando una riga è letta, viene eseguita l'operazione associata; le operazioni di stampa sono effettuate sullo standard output, e ogni operazione deve iniziare su una nuova riga.

Si assume che l'input sia inserito correttamente. Conviene scrivere le istruzioni di input in un file `in.txt` ed eseguire il programma redirigendo lo standard input.

Riga di input	Operazione
`r` n	**newRegister** (n)
`+` k *name*	**book**$(k, name)$
`-` k	**cancel**(k)
`m` $from$ to	**move**$(from, to)$
`p`	**printRegister**()
`f`	Termina l'esecuzione

Struttura dati

Per rappresentare il registro occorre usare un array `Register` allocato dinamicamente in quanto la dimensione è stabilita durante l'esecuzione del programma.

Sia n la dimensione di `Register`. Allora, in ogni istante del programma per ogni $0 \leq k < n$ deve valere la seguente proprietà:

- Se il posto k è prenotato da w, allora `Register[k]` è l'indirizzo a un vettore contenente w.
- Altrimenti, `Register[k]` vale `NULL` (indirizzo 0).

Esempio

INPUT

```
r 10
+ 1 Rossi
+ 3 Bianchi
p
m  1 5
p
+ 9 Verdi
p
- 3
p
r 20
+ 10 Mario
p
m 1 10
m 10 11
p
f
```

OUTPUT

```
REGISTER[0..9]:
1  --->  Rossi
3  --->  Bianchi

REGISTER[0..9]:
3  --->  Bianchi
5  --->  Rossi

REGISTER[0..9]:
3  --->  Bianchi
5  --->  Rossi
9  --->  Verdi

REGISTER[0..9]:
5  --->  Rossi
9  --->  Verdi

REGISTER[0..19]:
10  --->  Mario
move(1,10): errore

REGISTER[0..19]:
11  --->  Mario
```

Esercizio 2

a. L'obiettivo è di scrivere un programma per risolvere il problem delle n regine:

- Data una scacchiera $n \times n$, disporre n regine in modo tale che non si attacchino l'un l'altra.

Due regine si attaccano se si trovano sulla stessa riga oppure sulla stessa colonna oppure sulla stessa diagonale. Esempi di regine che si attaccano in una scacchiera 4×4:

```
- - - -
X - - X
- - - -
- - - -
```

Stessa riga

```
- - - -
- X - -
- - - -
- X - -
```

Stessa colonna

```
X - - -
- - - -
- - X -
- - - -
```

Stessa diagonale

Il problema ammette soluzione per $n \geq 4$.
Con la scacchiera standard 8×8, si hanno 92 soluzioni:

```
X - - - - - - -
- - - - X - - -
- - - - - - - X
- - - - - X - -
- - X - - - - -
- - - - - - X -
- X - - - - - -
- - - X - - - -
```

SOLUZIONE 1

```
X - - - - - - -
- - - - - X - -
- - - - - - - X
- - X - - - - -
- - - - - - X -
- - - X - - - -
- X - - - - - -
- - - - X - - -
```

SOLUZIONE 2

```
X - - - - - - -
- - - - - - X -
- - - X - - - -
- X - - - - - -
- - X - - - - -
- - - - - X - -
- - - - - - - X
- X - - - - - -
```

SOLUZIONE 3

...

Per lo svolgimento, leggere la traccia nel file `queen.c`.

b. Modificare il programma in modo che trovi una sola soluzione al problema. Per valore di n grandi, conviene anche modificare il formato dell'output. Ad esempio, per $n = 8$ indicando con $Q(r)$ la regina alla riga r ($0 \leq r < 8$) un possibile formato dell'output per la soluzione trovata (corrispondente alla Soluzione 1) è:

```
Q(0) = 0
Q(1) = 4
Q(2) = 7
Q(3) = 5
Q(4) = 2
Q(5) = 6
Q(6) = 1
Q(7) = 3
```

Nota
È anche possibile rappresentare la matrice $n \times n$ mediante un vettore Q di `int` di dimensione n tale che, per $0 \leq r < n$, $Q[r]$ indica la colonna in cui si trova la regine alla riga r (quindi $0 \leq Q[r] < n$).

Esercizio 3

Scrivere un programma `genera.c` che legge una parola w da standard input di lunghezza massima `WORD` prefissata, genera in modo casuale parole della stessa lunghezza di w fino a quando viene generata la parola w. Al termine dell'esecuzione, il programma stampa il numero di parole generate.

Il programma deve contenere:

- La funzione

  ```
  int randomRange(int a, int b)
  ```

  che, dati due interi a e b tali che $a \leq b$, genera in modo casuale un numero intero nell'intervallo $[a, b]$ (estremi inclusi). Ad esempio, se $a = -10$ e $b = 20$, la funzione deve restituire un intero n generato casualmente tale che $-10 \leq n \leq 20$.

- La funzione

  ```
  char* generateStringRand(int n)
  ```

  che restituisce un puntatore a un array contenente una parola generata casualmente composta da `n` caratteri (lettere minuscole dell'alfabeto). Usare la funzione `randomRange()`.

- La funzione

  ```
  int strEqual(char *w1, char *w2)
  ```

  che restituisce 1 se le parole `w1` e `w2` sono uguali, 0 altrimenti. Si può usare la funzione standard `strcmp()` che richiede l'inclusione di `<string.h>` (vedere la documentazione sui manuali).

La funzione `main()` deve compiere le seguenti operazioni:

(1) Viene letta in input una parola w di lunghezza massima `WORD` (da fissare).

(2) Viene calcolata la lunghezza l di w (si può usare la funzione standard `strlen()`).

(3) Vengono generate parole di lunghezza l fino a quando viene generata w.

Poiché il numero di parole da generare può essere elevato, conviene usare due contatori $countHigh$ e $countLow$ che si comportano come le lancette dell'orologio:

- $countLow$ è incrementato di uno ogni volta che una parola è generata.
- Quando $countLow$ raggiunge il valore 10^6, $countLow$ viene azzerato e $countHigh$ viene incrementato di uno.

Il numero di parole generate è quindi $countHigh \times 10^6 + countLow$.

Per controllare il processo di generazione, ad ogni incremento di $countHigh$ si può stampare un asterisco.

Un possibile messaggio di output è

```
****
Parole generate: 4 * 10^6 + 1235
```