

Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

mercoledì 6 novembre 20107

Registro di prenotazione

L'obiettivo dell'esercizio è scrivere un programma 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.

Funzionalità da implementare

Il programma deve implementare varie funzionalità. E' opportuno strutturare il programma in funzioni e commentare ciascuna funzione indicando chiaramente cosa fa e quali parametri usa.

- **newBook** (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.
- **printBook**()
Stampa il contenuto del registro (posti prenotati).

Notate che l'implementazione in C delle precedenti operazioni di alto livello può richiedere l'uso di parametri in più rispetto a quelli indicati. Riflettete quindi su quali siano i prototipi più opportuni da implementare!

Struttura dati

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

Sia n la dimensione di `book`. 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 `book[k]` è l'indirizzo a un vettore contenente w .
- Altrimenti, `book[k]` vale `NULL` (indirizzo 0).

Anche se avete la tentazione di definire `book` come una variabile globale, provate a definirla nel `main` e a passarla come argomento alle varie funzioni.

Formato di input e output

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.

Riga di input	Operazione
<code>b n</code>	<code>newBook(n)</code>
<code>+ k name</code>	<code>book(k, name)</code>
<code>- k</code>	<code>cancel(k)</code>
<code>m from to</code>	<code>move(from, to)</code>
<code>p</code>	<code>printBook()</code>
<code>f</code>	Termina l'esecuzione

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 reindirigendo lo standard input.

La lettura e l'interpretazione dei comandi può essere gestita con un ciclo contenente uno `switch`:

```
while( ( c = getchar() ) != 'f' ){
    switch(c){
        case 'b': // b n --> newBook(n)
            // ...
            break;
        case '+': // + k name --> book(k, name)
            //...
            break;
        case '-': // - k --> cancel(k)
            // ...
            break;
        case 'm': // m from to ---> move from to
            // ..
            break;
        case 'p': // p ---> printBook()
            // ...
            break;
    }
}
```

Esempio di funzionamento

INPUT

```
b 10
+ 1 Rossi
+ 3 Bianchi
p
m 1 5
p
+ 9 Verdi
p
- 3
p
b 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
```