Algoritmica – Prova di Laboratorio

Corso A e B

Appello del 09/09/2016

Istruzioni

Risolvete il seguente esercizio prestando particolare attenzione alla formattazione dell'input e dell'output. La correzione avverrà in maniera automatica eseguendo dei test e confrontando l'output prodotto dalla vostra soluzione con l'output atteso. Si ricorda che è possibile verificare la correttezza del vostro programma su un sottoinsieme dei input/output utilizzati. I file di input e output per i test sono nominati secondo lo schema: `input0.txt output0.txt input1.txt output1.txt ...` Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Una volta consegnata, la vostra soluzione verrà valutata nel server di consegna utilizzando altri file di test non accessibili. Si ricorda di avvisare i docenti una volta che il server ha accettato una soluzione come corretta.

Suggerimenti

Progettare una soluzione efficiente. Prestare attenzione ad eventuali requisiti in tempo e spazio richiesti dall'esercizio. In ogni caso, valutare la complessità della soluzione proposta e accertarsi che sia *ragionevole*: difficilmente una soluzione con complessità $\Theta\left(n^3\right)$ sarà accettata se esiste una soluzione semplice ed efficiente in tempo $\mathcal{O}(n)$.

Abilitare i messaggi di diagnostica del compilatore. Compilare il codice usando le opzioni `-g -Wall` di gcc:

```
gcc -Wall -g soluzione.c -o soluzione
```

risolvere *tutti* gli eventuali *warnings* restituiti dal compilatore, in particolar modo quelli relativi alle funzioni che non restituiscono un valore e ad assegnamenti tra puntatori di tipo diverso.

Provare la propria soluzione in locale. Valutare la correttezza della soluzione sulla propria macchina accertandosi che rispetti **tutti** gli input/output contenuti nel TestSet. Al fine di agevolare la verifica, si consiglia di posizionare la soluzione compilata e i file appartenenti al TestSet nella stessa directory e lanciare sotto tale directory lo script seguente:

```
for i in input* ; do
  ./soluzione < ${i} | diff -q - output${i##input}
done
```

Usare `Valgrind`. Nel caso in cui il programma termini in modo anomalo o non calcoli la soluzione corretta, è utile accertarsi che non acceda in modo scorretto alla memoria utilizzando `valgrind`:

```
valgrind ./soluzione < input0.txt
```

`Valgrind` eseguirà il vostro codice sull'input specificato (in questo caso, il file `input0.txt`), mostrando in output dei messaggi di diagnostica nei casi seguenti:

1. accesso (in lettura o scrittura) ad una zona di memoria non precedente allocata;
2. utilizzo di una variabile non inizializzata precedentemente;
3. presenza al termine dell'esecuzione del programma di zone di memoria allocate con `malloc` ma non liberate con `free` (*memory leak*).

Risolvere *tutti* i problemi ai punti 1. e 2. prima di sottoporre la soluzione al server.

Esercizio

Si vuole simulare la gestione di una coda di interi Q da N elementi ordinati secondo la politica LRU (*Last Recently Used*): in ogni momento, il primo elemento della lista è l'elemento letto più recentemente, il secondo è il secondo elemento letto più recentemente e così via.

In particolare, il programma deve leggere da input la lunghezza N della lista. Successivamente, il programma deve entrare in un ciclo in cui all'inizio di ogni iterazione va letto un intero e che indica una delle tre possibili operazioni da eseguire:

$e = 0$ (*terminazione*): termina il programma.

$e = 1$ (*accesso*): leggi da input un intero x ed aggiorna la coda seguendo i passi qui riportati:

1. cerca x in Q;
2. se x è già presente in Q, allora è necessario spostarlo in testa a Q;
3. se x *non* è già presente in Q, allora è necessario aggiungerlo in testa;
4. se x non era già presente in Q ed il suo inserimento nella coda fa si che la lunghezza della coda ecceda N, allora è necessario rimuovere l'elemento in coda a Q.

$e = 2$ (*stampa*): Stampa il contenuto di Q, partendo dall'elemento in testa.

L'input è formattato nel seguente modo. La prima riga è costituita dall'intero N che indica la dimensione della coda Q. Segue poi un numero indeterminato di comandi, formattati nel modo seguente:

$e = 0$: una riga contenente l'intero 0;

$e = 1$: una riga contenente l'intero 1, a cui segue una riga contenente l'intero x con cui aggiornare Q;

$e = 2$: una riga contenente l'intero 2.

L'output consiste negli output delle code richiesti dai comandi $e = 2$ in input. In particolare, l'output di Q deve essere formattato su una singola riga contenente gli interi in Q, dal primo all'ultimo, e terminata dal carattere \$. Ogni intero è separato dal successivo da un singolo spazio. Ad esempio, la stampa della coda $Q = \{1, 11, 2\}$ consiste nella riga `1 11 2 $`, mentre la coda vuota $Q = \{\}$ consiste nella riga contenente il singolo carattere `$`.

Note

- La coda può anche essere implementata mediante l'uso di un array.
- Se durante le vostre prove vi capitasse di non riuscire a interrompere il ciclo, per forzare la terminazione dell'esecuzione del programma si usi la combinazione di tasti `Ctrl+C`.

Esempio

Nota: il testo in verde è da intendersi come *commento* e dunque non fa parte dell'input.

Input

```
5  // N
1  // Comando e = 1: accesso
3  // Intero da accedere
1  // Comando e = 1: accesso
6  // Intero da accedere
2  // Comando e = 2: stampa
2  // Comando e = 2: stampa
1  // etc.
18
1
27
1
15
1
18
1
3
2
1
20
1
15
2
0
```

Output

```
6 3 $
6 3 $
3 18 15 27 6 $
15 20 3 18 27 $
```