Esercizi su strutture dati

Marco Alberti

Programmazione e Laboratorio, A.A. 2016-2017

Dipartimento di Matematica e Informatica - Università di Ferrara

Ultima modifica: 21 dicembre 2016

Liste collegate

Utilizzando il codice pubblicato a https://bitbucket.org/malbertife/programmazione2016/src/master/230_liste/, scrivere un programma in due versioni¹ che

1. crei una lista di interi;
2. richieda all'utente numeri interi positivi (fermandosi quando l'utente digita 0) e inserisca ognuno di essi nella lista in modo ordinato;
3. stampi la lista risultante.

---

¹Una con rappresentazione sequenziale e una con rappresentazione collegata della lista utilizzata)

Esercizio²

Implementare, per rappresentazione sequenziale e collegata, l'operazione di inserimento in coda a una lista di interi, con prototipo

```
void inserimento_coda(Lista *l, int numero);
```

Ad esempio, se $l = [1,4,2]$, l'inserimento in coda del numero 3 imposta l a $[1,4,2,3]$.

---

²Le soluzioni di questo e dei successivi esercizi su liste saranno pubblicate all'indirizzo https://bitbucket.org/malbertife/programmazione2016/src/master/230_liste/

Esercizio

Scrivere una funzione di prototipo

```
void inversa(Lista l1, Lista *l2);
```

la cui esecuzione faccia sì che la lista collegata *l2, inizialmente vuota, contenga gli stessi elementi della lista collegata l1, ma in ordine inverso.

Utilizzarla per scrivere un programma che richieda all'utente numeri positivi e inserisca i numeri in una lista collegata in modo ordinato dal minore al maggiore, e che, quando l'utente digita 0, stampi i numeri inseriti dal maggiore al minore.

Suggerimento

L'effetto desiderato si ottiene inserendo uno alla volta in testa a *l2 tutti gli elementi di l1.

Esercizio

Scrivere una funzione di prototipo

```
void concatena(Lista *l1, Lista l2);
```

che concateni l2 in coda a l1. Ad esempio, se $l1 = [1,2]$ e $l2 = [3,4]$, dopo la chiamata alla funzione $l1 = [1,2,3,4]$.

Esercizio

Scrivere una funzione di prototipo

```
void somma(Lista l1, Lista l2, Lista *l3);
```

la cui esecuzione abbia l'effetto di inserire in `l3`, inizialmente vuota, tutti gli elementi di `l1` seguiti da tutti gli elementi di `l2`.

Ad esempio, se `l1 = [1,2]` e `l2 = [3,4]`, dopo la chiamata alla funzione `l3 = [1,2,3,4]`.

Conteggio parole

Scrivere un programma da invocare come segue:

```
./conteggioParole __fileIn__ __fileOut__
```

che scriva nel file di testo `__fileOut__` tutte le parole distinte contenute nel file di testo `__fileIn__`, una per riga, in ordine alfabetico e con indicato il numero delle occorrenze in `__fileIn__`.

Per esempio, se il contenuto di `__fileIn__` è `to be or not to be`, il contenuto di `__fileOut__` dovrà essere

```
be 2
```

```
not 1
```

```
or 1
```

```
to 2
```

Suggerimenti

- Memorizzare ogni parola e il relativo numero di occorrenze in una struttura così definita:

```
typedef struct {  
    char parola[LUNGHEZZA_MAX];  
    int n_occorrenze;  
} frequenza;
```

- Costruire una lista collegata di nodi aventi come campo dato una `frequenza`, utilizzando una procedura simile a quella di inserimento ordinato già vista, ma che quando trova una parola uguale a quella da inserire incrementi il `n_occorrenze` della `frequenza` corrispondente, anziché creare un nuovo nodo.

Conteggio parole

- Non distinguere fra lettere maiuscole e minuscole (`Il` e `il` sono la stessa parola che compare due volte)
- Scrivere le parole nel file di uscita dalla più frequente alla meno frequente e, a parità di frequenza, in ordine alfabetico.

A questo scopo, costruire una seconda lista collegata con gli elementi della prima, per mezzo di una procedura di inserimento ordinato che usi come relazione di ordinamento quella per cui una `frequenza f1` precede un'altra `frequenza f2` se

1. `f1.n_occorrenze > f2.n_occorrenze`, oppure
2. `f1.n_occorrenze == f2.n_occorrenze` e `f1.parola` precede alfabeticamente `f2.parola`.

Pile

Notazione RPN

La Notazione Polacca Inversa (Reverse Polish Notation, RPN) è un linguaggio per operazioni aritmetiche in cui l'operatore segue gli operandi. Per esempio, l'espressione RPN $2\ 3\ +$ corrisponde all'espressione infissa $2 + 3$, mentre $3\ 4\ 1\ +\ /$ corrisponde a $3 / (4 + 1)$.

Una espressione RPN si può facilmente calcolare utilizzando uno stack di float e applicando le seguenti regole ai componenti dell'espressione letti da sinistra verso destra:

- se il componente è un operando, fare push dell'operando;
- se è un operatore ($+$, $-$, $\times$, $/$) applicarlo a un numero appropriato di elementi prelevati dallo stack (per esempio 2 per la somma) dall'ultimo al primo, e caricare il risultato sullo stack.

Il risultato è l'elemento di testa dello stack.

Esercizio

RPN

Scrivere un programma che calcoli e stampi il risultato dell'espressione RPN formata dagli argomenti della linea di comando. Per esempio, se l'eseguibile si chiama `rpn`, la chiamata

```
./rpn 3 4 1 + /
```

deve stampare `0.6`. Rappresentare con float tutti gli operandi e i risultati.

Implementare lo stack sia con array che con puntatori.

Nota

Per convertire una stringa in float, si può usare la funzione `strtof` (definita in `stdlib.h`). Per esempio, `strtof("3.4",NULL)` ha come risultato il float `3.4`.

Code

Coda con array

Implementare le operazioni `isQueueEmpty`, `isQueueFull`, `enqueue`, `dequeue` per l'ADT coda di interi, memorizzandone gli elementi in un array di interi.

Pratiche

Scrivere un programma che simuli la gestione di pratiche, ognuna rappresentata da una stringa di lunghezza massima 20, per mezzo di una coda. Il programma deve proporre all'utente un menu con le seguenti voci:

1. Creazione di una nuova pratica, che dovrà essere accodata;
2. Gestione di una pratica prelevata dall'inizio della coda: il programma deve stampare il titolo della pratica e chiedere all'utente se la pratica deve essere accodata;
3. Uscita dal programma.

Dopo ogni creazione o gestione, il programma deve stampare il contenuto della coda.