Laboratorio di algoritmi e strutture dati

Esercizi sull'uso delle pile*

Docente: Violetta Lonati

1 Calcolatrice in notazione postfissa

Un espressione è scritta in *notazione postfissa* quando gli operatori seguono gli operandi cui si applicano (nella usuale notazione infissa, invece, gli operatori appaiono in mezzo agli operandi cui si applicano). Ad esempio, l'espressione in notazione postfissa `5 3 -` equivale all'espressione in notazione infissa `5 - 3`. Combinando più operazioni si ottengono espressioni più complesse. Esempi:

- `5 3 - 2 *` è l'espressione in notazione postfissa che equivale all'espressione in notazione infissa `( 5 - 3 ) * 2`.
- `2 5 3 - *` è l'espressione in notazione postfissa che equivale all'espressione in notazione infissa `2 * ( 5 - 3 )`.

Con la notazione postfissa si eliminano i problemi dovuti alle parentesi e alla precedenza degli operatori (prima la divisione, poi l'addizione ecc.). Inoltre non c'è bisogno di annotare i risultati intermedi.

Nota: per semplicità, in questi esercizi consideriamo solo operatori binari, ovvero che si applicano ad una coppia di operandi. Inoltre scriviamo le espressioni separando operandi e operatori con degli spazi.

1.1 Valutazione di un'espressione in notazione postfissa

Per calcolare il valore di un espressione postfissa, si può usare un ciclo che esegue le seguenti azioni:

```
leggi un token (operatore o numero);
se il token è un numero
  inseriscilo nella pila;
se il token è un operatore
  estrai due valori dalla pila;
applica ad essi l'operatore;
inserisci il risultato nella pila;
```

*Ultimo aggiornamento: 3 novembre 2022 - 12:24:50

Alla fine la pila conterrà un solo valore, il risultato dell'espressone.

Scrivete una funzione `func valuta(espressione string) int` che implementa il precedente algoritmo, utilizzando una pila. La funzione riceve una espressione in notazione postfissa e restituisce il suo valore.

Ad esempio, se l'espressione è `5 3 - 2 *`, il valore restituito deve essere 4.

Per separare i token, si può usare la funzione `strings.Split`. Osservate che, per stabilire se `token` è un numero oppure un operatore, basta analizzare il suo primo carattere.

Notate che in questo caso la pila dovrà contenere dei numeri. Per implementarla, potete usare una *slice* eseguendo le *pop* e le *push* nella posizione più a destra. In alternativa, potete usare una lista concatenata eseguendo le *pop* e le *push* in testa.

1.2 Conversione da notazione infissa a notazione postfissa

La pila è utile anche per convertire un'espressione da notazione infissa (con parentesi che racchiudono ogni operazione) a notazione postfissa. In questo caso, la pila non conterrà i numeri, ma gli operatori, rappresentati da caratteri.

L'algoritmo è il seguente:

```
leggi un token;
se il token è un numero
  stampa il token;
se il token è un operatore
  inserisci il token in cima alla pila;
se il token è una parentesi aperta
  ignora il token
se il token è una parentesi chiusa
  estrai l'operatore in cima alla pila;
  stampalo;
```

Notate che gli operandi devono apparire nello stesso ordine in entrambe le notazioni (la cosa è rilevante nel caso di operazioni non commutative, come la sottrazione); inoltre si può osservare che le parentesi aperte non sono necessarie nella notazione postfissa (lo sarebbero invece nel caso di operazioni tra più di due operandi).

Scrivete una funzione `func converti(espressione string) string` che riceve una espressione in notazione infissa e restituisce l'espressione equivalente in notazione postfissa, usando l'algoritmo qui sopra (invece di stampare l'espressione convertita, dovrete costruire una stringa con l'espressione risultante).

Ad esempio, ricevendo per argomento la stringa `( ( 5 - 3 ) * 2 )` , la funzione deve restituire la stringa `5 3 - 2 *` .

2 Documenti html ben formati

Semplificando un po' le cose, in questo contesto chiamiamo documento html una sequenza di tag del tipo <a> (detti *tag di apertura*) oppure (detti *tag di chiusura*, dove a è una qualsiasi stringa di caratteri alfabetici.

Diciamo che un documento html è ben formato se i tag sono correttamente annidati, ovvero l'ordine dei tag soddisfa questi due criteri:

- per ogni tag di apertura esiste uno e un solo un tag di chiusura corrispondente
- se due tag di apertura compaiono in un determinato ordine, i corrispondenti tag di chiusura devono comparire nell'ordine opposto.

Ad esempio, la sequenza `<a> <b> </b> <c> <d> </d> </c> </a>` costituisce un documento html ben formato. Al contrario, la sequenza `<a> <b> </a> </c>` non lo è perchè il tag non è mai chiuso e il tag </c> non è mai aperto. Analogamente, la sequenza `<a> <b> </a> </b>` non è un documento html perchè il tag viene chiuso prima del tag .

Scrivete un programma che legga una sequenza di tag e stabilisca se costituisce un documento html ben formato oppure no. Per farlo, potete usare una pila e fare riferimento a questo ciclo:

```
leggi un tag t;
se t è un tag di apertura
  inserisci t nella pila;
se t è un tag di chiusura
  estrai il tag t2 dalla cima della pila;
  se t e t2 non corrispondono
    il documento non ben formato;
```

Il ciclo andrà ripetuto finché non ci sono più tag da leggere. Alla fine, affinché il documento sia ben formato, la pila deve essere vuota (pensate a degli esempi di input in cui restano tag nella pila alla fine dell'esecuzione).

Nota: che tipo di item contiene la pila questa volta?

Se il documento non è ben formato, il programma deve stampare un messaggio d'errore. Esempi:

- con la stringa
  `<a> <b> </b> </c>`
  il programma stampa il messaggio
  `errore in pos 4`
- con la stringa
  `<a> <b> </b> <c> <d> </d>`
  il programma stampa il messaggio
  `sono rimasti aperti i seguenti tag:  <a> <c>`