Algorithmen, Datenstrukturen und Programmieren II

SS 2001

1. `InfixToPostfixConverter`: Üblicherweise werden mathematische Ausdrücke in **infix**-Notation geschrieben, d.h. der Operator steht *zwischen* den Operanden, wie etwa in $3 + 4$ oder $7 * 9$. Für ein Computerprogramm ist es aber wesentlich leichter, Ausdrücke in **postfix**-Notation auszuwerten, d.h. der Operator steht *nach* den Operanden, obige Ausdrücke wären also $3\ 4\ +$ bzw. $7\ 9\ *$. Der Ausdruck $6 + 2 * 5$ wäre etwa $6\ 2\ 5\ *\ +$, $(6 + 2) * 5$ wäre hingegen $6\ 2\ +\ 5\ *$ (beachten Sie, daß keine Klammern notwendig sind).

   Schreiben Sie eine Applikation, welche einen einzulesenden infix-Ausdruck in einen postfix-Ausdruck umwandelt. Dabei sollen die Operatoren $+, -, *, /$ sowie Klammern zugelassen sein.

   Hier ein Beispiel für einen Algorithmus, der diese Aufgabe erfüllt. Definieren Sie zwei Variablen `StringBuffer infix` (in welcher der eingelesene Ausdruck gespeichert wird) und `StringBuffer postfix` sowie einen Stack.

   (a) pushen Sie eine linke Klammer '(' auf den Stack.

   (b) Hängen Sie eine rechte Klammer ')' ans Ende von `infix`.

   (c) Solange der Stack nicht leer ist, lesen Sie `infix` elementweise von links nach rechts:

   - Wenn das Element von `infix` eine Zahl ist, hängen Sie sie an `postfix` an.
   - Wenn das Element von `infix` eine linke Klammer ist, pushen Sie sie auf den Stack.
   - Wenn das Element von `infix` ein Operator ist:
     - Poppen Sie Operatoren (falls welche vorhanden) vom Stapel, solange diese gleiche oder höhere Priorität wie der aktuelle Operator haben, und hängen Sie diese gepoppten Operatoren ans Ende von `postfix`.
     - Pushen Sie das Element von `infix` (d.h. den Operator) auf den Stapel.
   - Wenn das Element von `infix` eine rechte Klammer ist:
     - Poppen Sie Operatoren (falls welche vorhanden) vom Stapel und hängen Sie sie ans Ende von `postfix`, bis eine linke Klammer oben am Stapel liegt.
     - Poppen Sie die linke Klammer vom Stapel und werfen Sie sie weg.

2. `PostfixEvaluator`: Ein Postfix-Ausdruck ist gleichzeitig der **Postorder Traversal** eines bestimmten Binärbaumes. Erzeugen Sie diesen Baum mittels folgendem Algorithmus und beachten Sie dabei die folgende **von rechts nach links**-Konvention : Die Anweisung "an ein Element anhängen" bedeutet "versuchen Sie es zunächst rechts anzuhängen, wenn dort schon besetzt ist, hängen Sie es links an".

   (a) Der äußerst rechte Operator bildet die Wurzel des Baumes. Erzeugen Sie damit einen neuen Knoten und legen Sie Ihn auf den Stack.

   (b) Lesen Sie `postfix` elementweise **von rechts nach links**:

   - Finden Sie den **letzten nicht vollbesetzten Operator**, indem Sie (falls notwendig) solange Knoten vom Stack poppen, bis oben auf dem Stack ein Knoten liegt, dessen rechte und/oder linke Kante auf `null` zeigt.
   - Erzeugen Sie einen neuen Knoten mit dem `postfix`-Element und hängen Sie Ihn an den oben auf dem Stack liegenden Knoten an.
   - Wenn das Element von `postfix` ein Operator ist, pushen Sie den eben erzeugten Knoten auf den Stack.

   Auswerten des Baumes (und damit des Postfix-Ausdrucks): Schreiben Sie eine rekursive Methode `int evaluate(node n)` nach folgendem Algorithmus:

   (a) Wenn `n.data` eine Zahl ist, liefert die Methode diese Zahl zurück.

   (b) Wenn `n.data` ein '+' ist, liefert die Methode `evaluate(n.left)+evaluate(n.right)` zurück.

   (c) Wenn `n.data` ein '-' ist, liefert die Methode `evaluate(n.left)-evaluate(n.right)` zurück, usw.

3. Schreiben Sie eine Applikation (als Unterklasse von `JFrame`), die einen Satz über ein `JTextField` einliest und dann folgende Daten ausgibt:

   a) Die Zahl der mehrfach vorkommenden Wörter (dabei soll Groß- und Kleinschreibung ignoriert werden).

   b) Die verschiedenen Wörter des Satzes in alphabetischer Reihenfolge.

   c) Die verschiedenen Wörter des Satzes in umgekehrter Reihenfolge.

Überlegen Sie zuerst, welche Datenstrukturen dem Problem angemessen erscheinen. Wahrscheinlich werden Sie die Klasse `StringTokenizer` verwenden wollen. Außerdem ist vielleicht die `Collections`-Methode `reverseOrder()` von Nutzen.

4. Verwenden Sie die Klasse `RandomFileTest` als Ausgangspunkt, um eine einfache Version einer Personalverwaltung mit GUI zu entwickeln (wieder als Unterklasse von `JFrame`): der User soll zunächst über Textfelder Name und Gehalt des Angestellten festlegen können. Das Programm soll dann die Nummer des jeweiligen Angestellten anzeigen und die Daten in einem File abspeichern.

   Es soll möglich sein, über die Personalnummer auf die einzelnen Angestellten zuzugreifen, ihre Daten anzeigen zu lassen und ihr Gehalt zu verändern. Außerdem soll eine Ausgabe des gesamten Datenfiles auf die Standardausgabe möglich sein. Bei Aufruf des Programmes soll die Zahl der derzeit im File abgespeicherten Angestellten abrufbar sein.

5. Modifizieren Sie das Programm `TableDisplay2.java` so, dass die Resultate der einzelnen Abfragen jeweils zusätzlich in Textdateien abgespeichert werden. Wenn der User den `Submit`-Button drückt, soll ein File namens `abfrageX.txt` angelegt werden, wobei `X` die Nummer der Abfrage ist. In dieses File soll zunächst der Abfrage-String geschrieben werden und danach als durch Kommata getrennte Liste das Resultat der Abfrage. Ein Blick auf die `showTable`-Methode des Programmes `MakeDB.java` ist eventuell von Nutzen...

6. Modifizieren Sie die Client-Server Applikation aus 4.4 so, dass der Client ein Textfile anfordern kann. Falls dieses File am Server nicht vorhanden ist, soll eine entsprechende Meldung an den Client gesendet werden. Ansonsten soll das File zeilenweise an den Client geschickt werden.