Informatik II (D-ITET)
FS 2009
Prof. Dr. F. Mattern
Philipp Bolliger (bolligph@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Verteilte Systeme
http://www.vs.inf.ethz.ch

Übungsserie Nr. 4

Ausgabe: 18. März 2009
Abgabe: 25./26. März 2009

1. Aufgabe: (4 Punkte) Umwandlung von Infix nach Postfix

In dieser Aufgabe müssen Sie ein Programm zur Umwandlung von Infix-Ausdrücken in Postfix-Notation schreiben, das *mehrziffrige* Operanden verarbeiten kann. Orientieren Sie sich dabei am Programm aus der Vorlesung (Postfixumwandlung mittels Stack, Folie 138 ff.).
Sie finden `InfToPost.java` (eine unseren Bedürfnissen angepasste Form des Programmes der Vorlesung) sowie die Hilfsklasse `Stack` auf der Vorlesungs-Webseite zum Herunterladen. Beachten Sie, dass nur *korrekte, vollständig geklammerte Ausdrücke* verarbeitet werden dürfen. Zum Übersetzen des Programms müssen dem Compiler die beiden Dateien `InfToPost.java` und `Stack.java` auf der Kommandozeile übergeben werden.

(1a) *(2 Punkte)* Fügen Sie im Code der Hilfsklasse `Stack.java` geeignete Kommentare ein, so dass die Methoden dieser Klasse ausführlich beschrieben sind. Geben Sie die so kommentierte Klasse Ihrem Tutor ab.

(1b) *(2 Punkte)* Testen Sie Ihr Programm mit den folgenden Eingaben und geben Sie die Ergebnisse sowie das vervollständigte Programm `InfToPost.java` Ihrem Tutor ab:

i) (5+((23+17)*9))

ii) (12+(5*((76+42)+19)))

2. Aufgabe: (8 Punkte) Auswertung von Syntaxdiagrammen

(2a) *(3 Punkte)* Geben Sie ein Syntaxdiagramm an, das als zulässige Ausdrücke Folgen von Multiplikationen erzeugt. Dabei sollen nur einzelne Ziffern (Terminalsymbole '0' bis '9') multipliziert werden dürfen; Klammern sowie der leere Ausdruck sind nicht vorgesehen. Beispielsweise sollen die folgenden Zeichenketten zulässige Ausdrücke nach Ihrem Syntaxdiagramm sein:

```
9
4*3
7*0*1
```

(2b) *(5 Punkte)* Laden Sie sich das Code-Fragment `Expr.java` von der Vorlesungs-Webseite herunter. Vervollständigen Sie die *rekursive* Java-Methode `int evalExprRec(int i)`, die einen (syntaktisch korrekten!) Ausdruck auswertet und das Resultat zurückgibt.

3. Aufgabe: (3 Punkte) Java-Bytecode

Betrachten Sie folgenden Ausschnitte eines Java-Bytecode-Programms:

```
Method int f(int, int, int)
   0 iload_0
   1 iload_1
   2 iadd
   3 iload_2
   4 idiv
   5 ireturn

Method int g(int, int)
   0 iload_0
   1 iload_1
   2 iconst_3
   3 invokestatic #27;  //Method f:<III>I
   6 ireturn
```

(3a) *(1 Punkt)* Die Methode `f()` hat drei Parameter, die den Variablen-Indizes (in den Vorlesungsunterlagen "vindex" genannt) 0, 1 und 2 zugeordnet sind. Nennen wir sie a (für den Index 0), b (1) und c (2). Welche Funktion berechnet die Methode `f()`?

(3b) *(1 Punkt)* Wie hoch wird der Operandenstack während der Ausführung von `f()` maximal?

(3c) *(1 Punkt)* Die Methode `g()` benötigt zwei Parameter (a, b) als Eingabe und ruft `f()` mit der Instruktion `invokestatic` auf. Welche Parameter übergibt sie an `f()`?

4. Aufgabe: (8 Punkte) Stackimplementierung der Ackermann-Funktion

Die Ackermann-Funktion $A(n, m)$ ist für alle $n, m \in \mathbb{N}_0$ definiert durch:

$$\begin{array}{lcl} A(0, m) & = & m + 1 \\ A(n + 1, 0) & = & A(n, 1) \\ A(n + 1, m + 1) & = & A(n, A(n + 1, m)) \end{array}$$

(4a) *(1 Punkt)* Berechnen Sie $A(1, 1)$ und $A(2, 2)$ "von Hand".

(4b) *(2 Punkte)* Beschreiben Sie (in Pseudocode) einen *iterativen* Algorithmus, der die Ackermann-Funktion mit Hilfe eines Stacks berechnet. Geben Sie den Algorithmus unter Verwendung der üblichen Stack-Operationen *push, pop, getTop*[1] und *empty* an. Am Anfang wird der Stack mit den Elementen n und m initialisiert. Wie lautet die Abbruchbedingung Ihres iterativen Algorithmus?

(4c) *(5 Punkte)* Implementieren Sie Ihren Algorithmus aus Teilaufgabe *b)* und geben Sie den entsprechenden Code Ihrem Tutor ab. Benutzen Sie dazu das Code-Fragment `Ackermann.java` und die Hilfsklasse `Stack`, welche Sie von der Vorlesungs-Webseite herunterladen können. Geben Sie die Folge der einzelnen Berechnungsschritte in Form von "Schnappschüssen" des Stacks an. (Bemerkung: Die Ackermann-Funktion wächst sehr schnell; z.B. ist $A(3, 3) = 61$, aber $A(4, 2)$ besitzt bereits 19729 Dezimalstellen!)

Summe: 23 Punkte

[1] `getTop` liefert das oberste Stack-Element, ohne es vom Stack zu entfernen.