TU Ilmenau, Fakultät IA
FG Telematik/Rechnernetze
Prof. Dr.-Ing. G. Schäfer
http://www.tu-ilmenau.de/telematik/pua

TECHNISCHE UNIVERSITÄT
ILMENAU

Programmierung und Algorithmen WS 23/24
Übungsblatt 4

Die Lösungen der Aufgaben sind bis zum 12.11.23, 23:59 Uhr abzugeben.
Die Besprechung der Aufgaben erfolgt in KW 46.

Aufgabe 1 (Wandlung von Programmierparadigmen) — 3 + 3 + 1 Punkte

(a) Formulieren Sie den folgenden applikativen Algorithmen als Java-Methode, welche sich am imperativen Programmierparadigma orientiert. Dies bedeutet insbesondere, dass ein rekursiver Methodenaufruf nicht erlaubt ist! Verwenden Sie ausschließlich solche elementare Operationen (u.a. Addition, Multiplikation), welche auch im applikativen Algorithmus verwendet werden. Gehen Sie davon aus, dass es sich bei `x` und `y` um ganze Zahlen handelt. Zusätzlich gelte $x \geq 1$. Der Rückgabewert der Algorithmen ist eine Fließkommazahl. `f(x,y) = if (y = 0) then 1`
`else if (y < 0) then (1.0)/f(x,-y)`
`else x · f(x,y-1) fi fi`

Hinweis: Um in Java bei der Division zweier `int`-Variablen ein gebrochenes Ergebnis zu erhalten, können Sie eine der Variablen mit 1.0 multiplizieren. Bsp: `float z = x*1.0/y`

(b) Formulieren Sie einen *applikativen Algorithmus*, welcher äquivalent zu folgendem imperativen Algorithmus ist:

```
COLLATZ:  var n : int;
          input n;
          while n > 1 do
               if n mod 2 = 0 then n := n / 2
               else n := (3 * n) + 1 fi;
          od;
          output n.
```

(c) Welche mathematische Funktion berechnet der erste Algorithmus?

Aufgabe 2 (Semantik imperativer Algorithmen) — 4 Punkte

Gegeben sei folgender imperativer Algorithmus:

```
Q:   var n, q : int;
     input n;
     q := 0;
     while n > 0 do
         q := q + n + n - 1;
         n := n - 1
     od
     output q.
```

Werten Sie den imperativen Algorithmus analog zum in der Vorlesung gezeigten Beispiel (Berechnung der Fakultät, Kapitel 4 Folie 44) auf der Eingabe $n = 2$ aus. Ebenfalls analog zur Vorlesung dürfen Sie die Abkürzung **while** β für die while-Schleife verwenden.

Aufgabe 3 (Semantik imperativer Ausdrücke) **4 Punkte**

Die Semantik imperativer Schleifenkonstrukte kann durch Funktionen auf Zuständen beschrieben werden. So etwa die *while*−Schleife:

$$[\![\texttt{while } B \texttt{ do } \alpha \texttt{ od}]\!](Z) = \begin{cases} \text{Z, falls Z(B)} = \texttt{false} \\ [\![\texttt{while } B \texttt{ do } \alpha \texttt{ od}]\!]([\![\alpha]\!](Z)), \text{ sonst} \end{cases}$$

Geben Sie die Semantik einer `for`$(\gamma;\ B;\ \delta)$ `do` α `od` Schleife entsprechend an. Die Semantik soll dabei der aus *Java* bekannten entsprechen.

Hinweis: γ, δ und α dürfen auch leere Anweisungsfolgen sein. Leere Anweisungsfolgen verändern den Zustand nicht.

Aufgabe 4 (Logische Programmierung) **2 + 5 Punkte**

Gegeben sei folgender deduktiver Algorithmus, welcher die Addition natürlicher Zahlen definiert. `ADD(X,Y,Z)` ist genau dann `true`, wenn `X + Y = Z` gilt.

Fakten: `SUC(n,n+1)` für alle $n \in \mathbb{N}_0$
Regeln: (1) $\Rightarrow$ `ADD(X,0,X)`
(2) `ADD(X,Y,Z)` $\wedge$ `SUC(Y,V)` $\wedge$ `SUC(Z,W)` $\Rightarrow$ `ADD(X,V,W)`

(a) Zeigen Sie mithilfe einer schrittweisen Auswertung, dass der Algorithmus `ADD(13,2,15)` korrekt bestimmt.

(b) Erweitern Sie das angegebene logische Programm um die folgenden Prädikate und die dazugehörigen Regeln, wobei es sich jeweils um Operationen auf den natürlichen Zahlen $\mathbb{N}_0$ handelt:

- `MULT(X,Y,Z)`: genau dann `true`, wenn `X` $\cdot$ `Y` = `Z`
- `FIB(X,Z)`: genau dann `true`, wenn `Z` die „`X`-te“ Fibonacci-Zahl ist. Es soll `FIB(0, 1)` und `FIB(1, 1)` gelten, siehe Definition der Fibonacci-Zahlen in Kapitel 4, Folie 16.

Aufgabe 5 (Backtracking und Sudokus) **6 Punkte**

Sudokus sind Rätsel, bei denen ein zweidimensionales Spielfeld so mit Zahlen ausgefüllt werden soll, dass bestimmte Regeln gelten. Für den Fall eines Spielfeldes der Größe 9×9 gilt folgendes:

- Es dürfen nur Zahlen zwischen 1 bis 9 auf dem Spielfeld verwendet werden.
- In jeder Reihe des Spielfeldes darf jede Zahl nur einmal vorkommen.
- In jeder Spalte des Spielfeldes darf jede Zahl nur einmal vorkommen.
- Das Spielfeld lässt sich in neun kleinere 3×3 - Spielfelder aufteilen. Innerhalb dieser kleineren Felder darf jede Zahl nur einmal vorkommen.
- Das Spielfeld ist zu Beginn nicht leer. Vorgegebene Zahlen dürfen *nicht* verändert werden.

Ergänzen Sie das in `Sudoku.java` vorgegebene Programm so, dass beliebige 9×9-Sudokus mithilfe von *Backtracking* gelöst werden können. Sie erhalten zusätzlich mit `sudoku.txt` ein lösbares Sudoku-Feld als Textdatei. Die Zahl 0 in der Datei entspricht einem leeren Feld.

Das allgemeine Prinzip des Backtrackings sieht vor, zunächst eine Teillösung zu konstruieren (hier: ein teilweise gelöstes Sudoku) und diese dann weiter auszubauen. Stellt man dabei fest, dass eine

Teillösung nicht zu einer Gesamtlösung ausgebaut werden kann (hier: keine gültige Zahl möglich für ein leeres Feld), wird der letzte Schritt der Teillösung zurückgenommen und stattdessen Alternativen probiert (hier: vorher gesetzte Zahlen ändern). Backtracking lässt sich dabei oft elegant *rekursiv* implementieren.

Hinweis: Für eine rekursive Implementierung bietet sich eine Hilfsmethode `boolean solveRec(int x, int y)` an, welche versucht, einen gültigen Wert für das Feld an Position (x, y) zu finden, und anschließend durch einen rekursiven Aufruf eine Lösung für ein nächstes Feld sucht.

Folgende Hilfsmethoden stehen Ihnen zur Verfügung:

- `display()` zeigt den aktuellen Zustand des Sudoku-Feldes an.
- `isEmpty(x, y)` überprüft, ob das Feld in Position (x, y) leer ist oder eine Nummer enthält.
- `writeNumber(x, y, val)` schreibt eine Nummer `val` auf Position (x, y).
- `clearNumber(x, y)` löscht eine Nummer auf Position (x, y).
- `checkValidNumber(x, y, val)` überprüft, ob ein Setzen der Nummer `val` auf Position (x, y) gegen Sudoku-Regeln verstößt. Ein Aufruf dieser Methode verändert das Feld selbst *nicht*!

Hinweis: Die oben genannten Methoden berücksichtigen nicht, ob es sich bei einer Zahl auf Position (x, y) um eine durch das importierte Feld fest vorgegebene Zahl handelt.