PRG1x & ADE1x	**Einf. i. d. Programmierung (int. LVA)** **Üb. zu Element. Alg. u. Datenstrukt.**	**WS 13/14, Übung 8**

Abgabetermin: Sa, 21.12.2013

☐ **Gr. 1,** DI (FH) G. Horn, MSc

☐ **Gr. 2,** J.-P. Haslinger, MSc **Name** _____ **Aufwand in h** _____

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

. Das Sierpinski-Dreieck (4 + 2 + 4 Punkte)

Das Sierpinski-Dreieck ist ein 1915 von Wacław Sierpiński beschriebenes Fraktal. Es lässt sich, wie unter http://de.wikipedia.org/wiki/Sierpinski-Dreieck beschrieben, durch einen iterativen Prozess konstruieren, wobei Fläche des Dreiecks von Iteration zu Iteration abnimmt während die Gesamtlänge aller Kanten immer weiter steigt. Die nachstehende Abbildung zeigt das Sierpinski-Dreieck zu Beginn und nach den ersten drei Iterationen.

a) Entwickeln Sie eine rekursive Funktion, die die Fläche des Sierpinski-Dreiecks für eine gegebene Anzahl von Iteration ermittelt. (Annahme: Fläche des Basisdreiecks = 1)

b) Implementieren Sie auch eine iterative Lösung für Ihre Funktion.

c) Entwickeln Sie nun auch eine rekursive sowie eine iterative Variante einer Funktion, die die Gesamtlänge aller Kanten im Sierpinski-Dreieck für eine gegebene Anzahl von Iterationen berechnet. (Annahme: Seitenlänge des Basisdreiecks = 1)

2. Erreichbarkeit von Feldern (3 + 8 + 3 Punkte)

Gegeben sei ein Spielfeld in der Größe 50 x 20, dessen einzelne Felder entweder frei oder eine Wand sein können. Gesucht ist nun ein Algorithmus, der für zwei Felder A und B auf dem Spielfeld ermittelt, ob eine Figur von A nach B ziehen kann, ohne dabei durch eine Wand zu laufen. Die Figur darf sich dabei pro Zug jeweils entweder ein Feld nach oben, nach unten, nach links oder nach rechts bewegen.

a) Entwerfen Sie eine geeignete Datenstruktur für das Spielfeld und bauen Sie zu Testzwecken eine Prozedur zum Setzen von Wänden sowie zum Ausgeben des Spielfelds mit seinen Wänden auf dem Bildschirm.

b) Implementieren Sie eine rekursive Funktion

```
FUNCTION PathExists(ax, ay, bx, by: INTEGER): BOOLEAN;
```

die prüft, ob das Feld *bx*/*by* von *ax*/*ay* aus in *steps* Schritten erreicht werden kann und testen Sie diese mit aussagekräftigen Beispielen.

c) Bauen Sie (aufbauend auf die Implementierung von *PathExists*) eine neue, „bessere“ Funktion

```
FUNCTION LengthOfShortestPath(ax, ay, bx, by: INTEGER): INTEGER;
```

die nun sogar die Länge des kürzesten Pfades zwischen *ax/ay* und *bx/by* ermittelt. Existiert kein möglicher Pfad, so ist -1 zurückzuliefern.

Erstellen Sie (mindestens) ein möglichst geschicktes Spielfeld und testen Sie sowohl *PathExists* als auch *LengthOfShortestPath* ausführlich.

Beispiele: („.“ = frei / „#“ = Wand)

```
. . . . . . . . . . . . . .
. . . . # . . . . . . . . .
. . . # # # # # . . . . . .
. . # # # . # . . . . . . .
. . . . . . # . . . # # # #
# # # # . . . . . # . . . .
. . . # # # # # . . # . . .
. . . . . . . . . . # . . .
```

ax	ay	bx	by	PathExists	LengthOfShortestPath
2	1	5	1	TRUE	3
5	1	5	1	TRUE	0
2	1	5	2	FALSE	-1
5	2	5	2	FALSE	-1
6	4	1	7	TRUE	16
1	7	6	4	TRUE	16
6	4	6	2	TRUE	12
1	1	14	8	FALSE	-1
12	6	14	8	TRUE	4
11	6	14	8	FALSE	-1