TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Praktikum Grundlagen der Programmierung
Prof. Dr. Helmut Seidl

WS 2007/2008
Aufgabenblatt 4
9. November 2007

Abgabe: Montag, 19.11.07 um 8 Uhr per Mail an den jeweiligen Tutor

Praktikum Grundlagen der Programmierung

Aufgabe 22 (Ü) Fibonacci-Folge

Die Folge der Fibonacci-Zahlen ist definiert als:

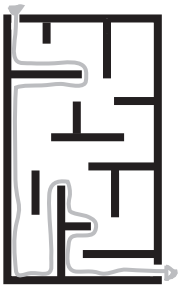
$$n_0 = 0 \qquad n_1 = 1 \qquad n_i = n_{i-1} + n_{i-2} \text{ für } i > 1$$

Schreiben Sie ein Programm, das nach Eingabe des Index i die Fibonaccizahl n_i berechnet und ausgibt. Überlegen Sie sich sowohl eine rekursive, als auch eine iterative Lösung. Vergleichen Sie die beiden Lösungen hinsichtlich ihrer Laufzeit.

Aufgabe 23 (Ü) Labyrinth

Um in einem Labyrinth vom Eingang zum Ausgang zu kommen, tastet man sich so an den Wänden entlang, dass man immer zu seiner rechten Seite eine Wand spürt, siehe Abbildung. Implementieren Sie das Durchwandern des Labyrinths in Java unter Zuhilfenahme der Klasse `Maze`, die Ihnen auf der Übungs-Website zu Verfügung steht. Sie bietet folgende Methoden:

- `boolean[][] generateMaze(int width, int height)`: Erzeugt ein zufälliges Labyrinth, repräsentiert als ein boolesches 2-dimensionales Array mit `width` Spalten und `height` Zeilen. Hat dieses Array an Stelle $[x][y]$ den Wert `true`, befindet sich im Labyrinth bei den Koordinaten (x,y) eine Wand.
- `draw(int x, int y, boolean[][] maze)`: Gibt eine ASCII-Darstellung des Labyrinths `maze` mit der Person (markiert als I) bei den Koordinaten (x,y) aus.



Der Eingang befindet sich links oben im Labyrinth bei den Koordination $(1,0)$, der Ausgang rechts unten im Labyrinth bei den Koordinaten `(width-1,height-2)`. Ein Labyrinth soll demnach nicht kleiner als 3×3 Zellen sein. Benutzen Sie die `draw`-Methode, um den Weg der Person durchs Labyrinth zu illustrieren. Wenn im generierten Labyrinth kein Weg zum Ausgang existiert - was durchaus vorkommen kann - soll Ihr Programm dies bemerken und sich beenden.

Aufgabe 24 (Ü) Quadrat-Fraktale

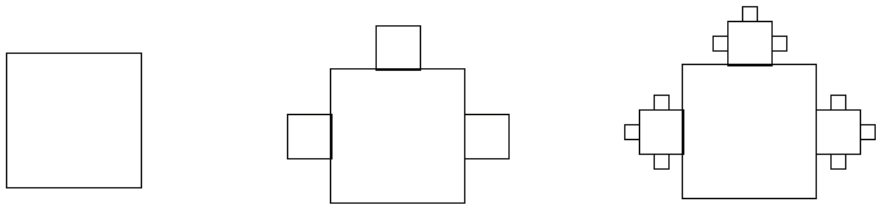
Fraktale sind selbstähnliche geometrische Formen, die durch wiederholte Anwendung einer Erzeugungsregel auf eine Ausgangsform entstehen. Für das Fraktal, das in dieser Aufgabe gezeichnet werden soll, gelte:

- Ausgangsform: Quadrat.
- Erzeugungsregel: füge in der Mitte jeder Seite s des Fraktals ein neues Quadrat der Länge $l/3$ hinzu, wobei l die Länge von s bezeichnet.

Dabei soll das Fraktal nur nach außen wachsen (siehe Abbildung unten). Zum Zeichnen der Quadrate stellt die Klasse `SquareFractal` von der Übungs-Website folgende Methoden bereit:

- `initFrame()`: Initialisiert die Zeichenfläche mit 500×500 Pixel.
- `drawRectangle(int x, int y, int width, int height)`: Zeichnet an der Position (x,y) ein Rechteck der Breite `width` und Höhe `height`.

Ihr Programm soll dabei die Anzahl der zu zeichnenden Iterationsschritte des Fraktals von der Kommandozeile einlesen. Die folgende Abbildung zeigt die Ausgangsform und die ersten zwei Iterationen des Fraktals:



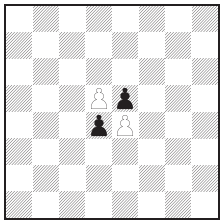
Aufgabe 25 (H) Schatzkiste (6 Punkte)

Ein Schatzgräber hat eine Schatztruhe voller wertvoller Schätze entdeckt. Leider kann er die Truhe nicht alleine schleppen und muss so die Schätze in seinem Rucksack verstauen. Jedes der gefundenen Schmuckstücke hat ein bestimmtes Gewicht und einen bestimmten Wert. Der Schatzgräber möchte seinen Rucksack nun so packen, dass der Rucksackinhalt das Maximalgewicht, das der Schatzsucher schleppen kann, einhält und er durch die mitgenommenen Schmuckstücke einen maximalen Wert erzielt.

Implementieren Sie ein optimales Packen des Rucksacks unter Verwendung *rekursiver Funktionsaufrufe*. Geben Sie am Ende Ihres Programms den Inhalt des gefüllten Rucksacks, sowie dessen tatsächliches Gewicht und seinen Wert aus.

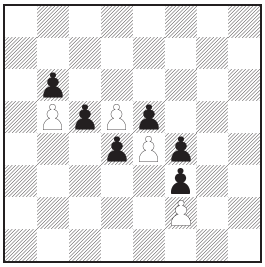
Das Gewicht und den Wert der einzelnen Schmuckstücke der Schatztruhe, liefert Ihnen die Methode `int[][] getSchatz(int n)` der Klasse `Schatz`, wobei in der ersten Spalte des $n \times 2$ Arrays das Gewicht des Schmuckstücks und in der zweiten Spalte dessen Wert abgelegt ist. Der Parameter `int n` gibt die Anzahl der gefundenen Schmuckstücke der Schatztruhe an. Schreiben Sie ein `main`-Programm, welches Ihr Packen des Rucksacks überprüft.

Aufgabe 26 (H) Legespiel **(12 Punkte)**

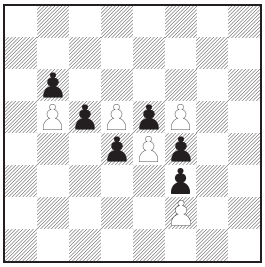


Auf einem Schachbrett mit 8×8 Feldern soll folgendes Legespiel simuliert werden. Abwechselnd legen der Spieler und der Computer Spielsteine, deren Seiten unterschiedlich gefärbt sind (weiss und schwarz). Der Spieler legt seinen Stein immer mit der weissen Seite nach oben, wohingegen der Computer seinen Stein mit der schwarzen Seite nach oben ablegt. Zu Beginn des Spiels liegen 2 schwarze und 2 weisse Steine in der nachfolgenden Anordnung auf dem Schachbrett.

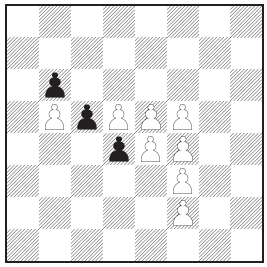
Jeder der beiden Spieler muss seinen Stein auf ein leeres Feld legen, welches entweder horizontal, diagonal oder vertikal an ein schon belegtes Feld angrenzt. Wird ein Stein im Schachbrett plaziert, so werden alle gegnerischen Steine umgedreht, die unmittelbar zwischen dem neuen Spielstein und einem schon gelegten Spielstein der eigenen Farbe liegen. Für das Spiel stehen den beiden Spielern unbegrenzt viele Spielsteine zur Verfügung.



a)



b)



c)

Als Beispiel legt der Spieler ausgehend von dem Spielfeld in Abbildung *a*) seinen weissen Spielstein wie in Abbildung *b*), so ergibt sich nach Umdrehen das Spielfeld aus Abblidung *c*).

Schreiben Sie ein Progrmm, welches dieses Legespiel simuliert.

a) Definieren Sie eine Methode `legalerZug`, welche prüft ob der Spielstein an diese Position plaziert werden darf d.h. der Platz noch frei ist und ein angrenzendes Nachbarfeld schon belegt ist.
b) Definieren Sie eine Methode `dreheUm`, die abhängig von dem plazierten Stein alle Steine in diagonaler, vertikaler und horizontaler Richtung gemäss obiger Regel umdreht (siehe Beispiel).
c) Definieren Sie die Methode `computerZug`, die die Position für den Spielstein des Computers bestimmt, an der maximal viele Steine umgedreht werden.

Für Ihr Legespiel steht Ihnen die Klasse `Legespiel` zur Verfügung, die folgende Methoden bereitstellt:

- **`void`** `initLegespiel(`**`int`**`[][] spielFeld)`: Öffnet ein Fenster, in dem die Simulation gestartet wird.
- **`void`** `updateLegespiel(`**`int`**`[][] spielFeld)`: Aktualisiert und visualisiert das Spielfeld. Hierbei soll angenommen werden, dass die Zahl 0 ein noch nicht belegtes Feld, die Zahl -1 einen schwarzen und die Zahl 1 einen weissen Spielstein auf dem Schachbrett symbolisiert.
- **`int`**`[] getSpielerZug()`: Liefert die Koordinaten des beim letzten Klick des Spielers gesetzen Steins, wobei an Stelle 0 die x-Koordinate und an Stelle 1 die y-Koordinate abgespeichert ist.