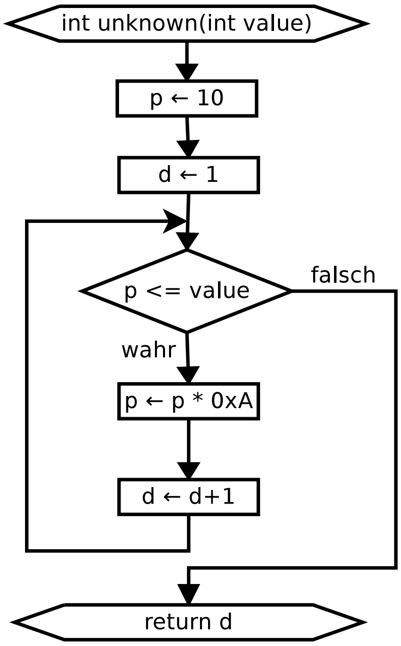
3. Übung

Abgabe bis 13.11.2017, 10:00 Uhr

Einzelaufgabe 3.1: Ablaufdiagramm: AD2J — 4 EP

In der Vorlesung haben Sie zu jeder Kontrollstruktur in Java die zugehörige Darstellung als Programmablaufplan (PAP) kennengelernt.

a) Laden Sie die vorgegebene Datei `Unknown.java` von der Übungsseite und ergänzen Sie die darin vorbereitete Methode `int unknown(int value)` um den folgenden Ablauf:



Testen Sie Ihre Implementierung gründlich und geben Sie Ihre Lösung als `Unknown.java` über das EST ab.

b) Überlegen Sie weiterhin was die Methode allgemein (abhängig von der Eingabe) berechnet. Geben Sie Ihre Lösung als `Unknown.pdf` über das EST ab.

Einzelaufgabe 3.2: Ablaufdiagramm: J2AD — 15 EP

Um die Ausführung eines komplexen Programms leichter nachvollziehen zu können, hilft es oft, den sogenannten Kontrollflussgraphen zu zeichnen.

a) Laden Sie `R.java` von der Webseite der AuD-Übungen und stellen Sie die darin definierte Methode `int[] r(int[] content)` graphisch in einem Ablaufdiagramm dar. Verwenden Sie dabei eine vergleichbare Darstellung wie in der Abbildung zu Aufgabe 3.1.

b) Führen Sie den Methodenaufruf `int[] result = r(test);` wie in der `main`-Methode mit `int[] test = {5, 2, 3};` gedanklich in einem Schreibtischlauf aus und geben Sie tabellarisch in folgender Form an, welche Werte die jeweiligen Variablen an den im Code mit `// 1) --> cur j ra result <--` bzw. `// 2) --> cur j ra result <--` kommentierten Stellen (also *nach* der Ausführung der jeweiligen Zeile) jeweils annehmen:

cur	j	ra	result
?	?	?	? ? ? ?
...	...	...	...

c) Formulieren Sie in *einem Satz*, was die Methode berechnet.

Geben Sie Ihre Lösung als `R.pdf` über das EST ab.

Einzelaufgabe 3.3: Lotteriegewinn **4 EP**

In dieser Aufgabe berechnen Sie die Wahrscheinlichkeit eines Lotteriegewinns. Bei einer Lottoziehung werden jeweils m Zahlen aus einer Menge von n Zahlen gezogen (z.B. für Deutschland gilt z.B. „*6 aus 49*“, also $m = 6$ und $n = 49$). Je mehr Zahlen richtig getippt wurden, umso höher ist die Gewinnsumme. Im folgenden gehen wir davon aus, dass nur derjenige gewinnt, der alle Zahlen (ohne Zusatzzahl) korrekt getippt hat.
Die Wahrscheinlichkeit für einen Gewinn hängt davon ab, wie viele verschiedene Möglichkeiten es gibt, um m Zahlen aus einer Menge von n Zahlen zu ziehen. Diese Anzahl berechnen Sie mit dem Binomialkoeffizienten (1):

$$\binom{n}{m} = \frac{n!}{m! \cdot (n-m)!} \tag{1}$$

Der Binomialkoeffizient kann unter Verwendung der Fakultätsfunktion (2) berechnet werden:

$$k! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k = \prod_{i=1}^{k} i \quad \text{mit} \quad 0! := 1 \tag{2}$$

Um die Wahrscheinlichkeit p („Gewinnchance“) zu bestimmen, mit der Sie genau die richtigen Zahlen tippen, müssen Sie den reziproken Wert des Binomialkoeffizienten (1) berechnen:

$$p = \frac{1}{\binom{n}{m}} = \frac{1}{\frac{n!}{m! \cdot (n-m)!}} \tag{3}$$

Laden Sie die Datei `Lotterie.java` herunter und ergänzen Sie die vorgegebenen Methoden. Sie dürfen dabei ohne eigene Prüfung annehmen, dass für alle Eingaben stets $0 < m \leq n$ gilt. Geben Sie Ihre Lösung als `Lotterie.java` über EST ab.

Gruppenaufgabe 3.4: Funktionen darstellen nach ASCII-Art **22 GP**

In dieser Aufgabe sollen Sie den Graphen einer Funktion mit Legende und Koordinatensystem zeichnen. Laden Sie dazu die Datei `FunctionPlotter.java` von der AuD-Homepage herunter und machen Sie sich mit den Kommentaren vertraut.
Die graphische Darstellung der Funktion samt Koordinatensystem und weiterer Daten steht in der Variablen `plottingArea` der Klasse `FunctionPlotter`. Um die Darstellung auf der Standardausgabe auszugeben, verwenden Sie die bereits implementierte Methode `printPlottingArea()`.
Die Koordinaten der `plottingArea` orientieren sich an der mathematischen Notation, d.h. die erste Dimension (`x`-Koordinate) läuft von links nach rechts, die zweite Dimension (`y`-Koordinate) läuft von unten nach oben.
Von den Methoden, die Sie implementieren sollen, dürfen nur `newPlottingArea` und `plotChar` das Feld `plottingArea` verändern – falls Sie in einer der anderen Methoden `plottingArea` modifizieren müssen, verwenden Sie stattdessen eine dieser beiden Methoden.

a) Die Methode `newPlottingArea` soll die Variable `plottingArea` geeignet initialisieren. Es soll als neues zweidimensionales Feld von `char`-s mit `width` Zeilen und `height` Spalten angelegt werden. Dieses Feld *muss vollständig* mit **Leerzeichen (' ')** initialisiert werden.

b) Die Methode `plotChar` soll zuerst die übergebenen Koordinaten (`x`, `y`) prüfen. Liegen sie in den Grenzen des `plottingArea`-Felds, dann soll sie an dieser Stelle das als Parameter `c` übergebene Zeichen eintragen.

c) Die Methode `plotHorizontalLine` (sic!) soll eine horizontale Linie vom Startpunkt (`xStart`, `y`) bis zum Endpunkt (`xEnd`, `y`) (jeweils inklusive) zeichnen. Die Linie soll dabei mit dem im Parameter `c` übergebenen Zeichen gezeichnet werden. Sie dürfen davon ausgehen, dass `xStart` $\leq$ `xEnd` ist. Beachten Sie, dass zum Zeichnen die Methode `plotChar` verwendet werden muss!

d) Analog dazu soll die Methode `plotVerticalLine` eine vertikale Linie vom Startpunkt (`x`, `yStart`) bis zum Endpunkt (`x`, `yEnd`) (jeweils inklusive) zeichnen. Die Linie soll dabei wieder mit dem Zeichen in `c` gezeichnet werden. Auch hier ist zu beachten, dass `yStart` $\leq$ `yEnd` ist und dass zum Zeichnen die Methode `plotChar` verwendet werden muss!

e) Die Methode `plotBox` soll ein Rechteck so zeichnen, wie es der Kommentar der Methode vorgibt.

f) Die Methode `plotString` soll die übergebene Zeichenkette zeichnen. Das erste Zeichen der Zeichenkette `s` soll dabei an die Stelle (`x`, `y`) gezeichnet werden, während die restlichen Zeichen entsprechend rechts daneben stehen sollen.

g) Die Methode `plotFunction` soll den Graphen der Funktion `functionToPlot` ausgeben. Dazu soll `functionToPlot` für jedes x zwischen `xStart` und `xEnd` (jeweils inklusive) ausgewertet werden. Diese Werte sollen dann mit dem Zeichen `'.'` gezeichnet werden.
Der Nullpunkt der Funktionsdarstellung stimmt dabei nicht notwendigerweise mit dem Nullpunkt des `plottingArea`-Feldes überein – daher werden die Koordinaten des Nullpunkts so übergeben, dass sich der Nullpunkt genau an der Stelle (`xOrigin`, `yOrigin`) im `plottingArea`-Feld befindet.

Geben Sie Ihre Lösung als `FunctionPlotter.java` über EST ab.

Gruppenaufgabe 3.5: Newton-Iterationen — 15 GP

Das sogenannte *Newton-Verfahren* ist eine einfache, aber recht effiziente Methode zur rechnerischen Lösung von Gleichungen. Generell ist das Ziel des Newton-Verfahrens, die Nullstelle einer Funktion f zu finden. Dazu startet man an einem beliebigen Punkt y_{alt} und berechnet wiederholt:

$$y_{neu} \leftarrow y_{alt} - \frac{f(y_{alt})}{f'(y_{alt})} \tag{4}$$

Sobald $y_{neu} \approx y_{alt}$ ist, hat man eine Näherung y_{neu} für die Nullstelle gefunden – ansonsten ersetzt man y_{alt} durch y_{neu} und fängt von vorne mit (4) an.
Beispiel: Für die Berechnung der Quadratwurzel $\sqrt{x}$ wird die Funktion $f(y) = y^2 - x$ angesetzt. Deren (positive) Nullstelle ist $y_{loesung} = \sqrt{x}$. Die Ableitung (nach y) von f ist $f'(y) = 2 \cdot y$.
Sie sollen im Folgenden Schritt für Schritt ein Programm entwickeln, welches mit diesem Verfahren $\sqrt[n]{x}$ für beliebige n und x näherungsweise bestimmt. Laden Sie zunächst den Rumpf der Klasse `NewtonIteration.java` von der Homepage der Übungen.

a) Implementieren Sie die Methode `power(double x, int n)` so, dass sie x^n zurückgibt. Sie dürfen davon ausgehen, dass `n` immer größer oder gleich 0 ist.

b) Ergänzen Sie die Methoden `fun` sowie `funDeriv` dahingehend, dass diese die parametrisierte Funktion $f_{n,x}(y) = y^n - x$ bzw. deren Ableitung (nach y) berechnen. Sie sollten dazu die `power`-Methode aus der vorangehenden Teilaufgabe verwenden.

c) Passen Sie `newtonStep` unter Verwendung der vorangehenden Methoden so an, dass sie für gegebene Werte von n, x und y_{alt} den „neuen" Wert y_{neu} nach Formel (4) zurückgibt.

d) Implementieren Sie nun die Methode `approxRoot` so, dass sie eine Näherung für $\sqrt[n]{x}$ berechnet und zurückgibt. Sie sollen die Berechnung abbrechen, sobald y_{neu} und y_{alt} näher als `EPSILON` beieinander liegen. Sie dürfen zur Implementierung die Methode `Math.abs` aus der Java-API sowie alle Methoden aus den Teilaufgaben a), b) und c) verwenden. Als Startwert für y_{alt} **müssen Sie 1** verwenden.

WICHTIG: Sie dürfen außer `Math.abs` und den zu implementierenden Methoden in dieser Aufgabe **keine** andere Klasse oder Methode aus der Java-API verwenden!
Geben Sie Ihre Lösung als `NewtonIteration.java` über EST ab.

23 EP + 37 GP = 60 Punkte