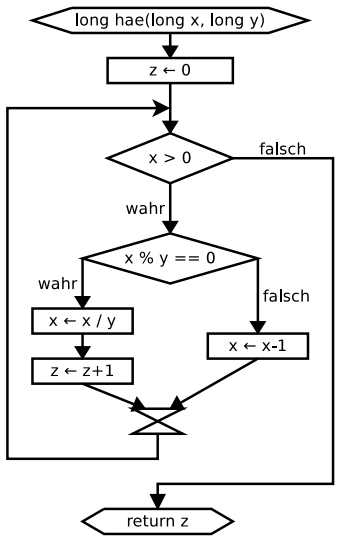
FAU, Informatik 2, AUD-Team
aud@i2.cs.fau.de

3. Übung

Abgabe bis 14.11.2016, 10:00 Uhr

Einzelaufgabe 3.1: AD2J — 10 EP

In der Vorlesung haben Sie zu jeder Kontrollstruktur in Java die zugehörige Darstellung als Ablaufdiagramm kennengelernt. Betrachten Sie nun folgendes „Programm“:



a) Führen Sie `hae(666, 5)` in einem „Schreibtischlauf“ aus und geben Sie die Belegung von `x` und `z` nach *jedem einzelnen* Schleifendurchlauf tabellarisch wie folgt an:

Schleifendurchlauf	`x`	`z`
1	...	...

b) Formulieren Sie in einem Satz genau, was die Methode *allgemein* (unabhängig von konkreten Werten für `x` bzw. `y`) berechnet – oder geben Sie eine mathematische Formel dafür an.

c) Laden Sie die vorgegebene Datei Hae.java von der Übungsseite und ergänzen Sie die darin vorbereitete Methode `long hae(long x, long y)` um den obigen Ablauf.

Geben Sie Ihre Lösung als `Hae.pdf` bzw. `Hae.java` ab.

Einzelaufgabe 3.2: J2AD — 13 EP

Betrachten Sie nun die Methode `Hokus.pokus` im folgenden Java-Programm:

Hokus.java

```
1  public class Hokus {
2      public static void pokus(int s, char in[], char out[]) {
3          for (int i = 0; i < in.length; ++i) {
4              char ch = in[i];
5              if (ch >= 'A' && ch <= 'Z') {
6                  ch += s;
7                  if (ch > 'Z')
8                      ch -= 26;
9                  if (ch < 'A')
10                     ch += 26;
11                 out[i] = ch;
12             } else if (ch >= 'a' && ch <= 'z') {
13                 ch += s;
14                 if (ch > 'z')
15                     ch -= 26;
16                 if (ch < 'a')
17                     ch += 26;
18                 out[i] = ch;
19             } else {
20                 out[i] = ch;
21             }
22         }
23     }
24
25     public static void main(String[] args) {
26         char[] in = { 'A', 'u', 'D', ' ', 'i', 's', 't', ' ', 'S', 'u', 'p', 'i' };
27         char[] out = { '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?' };
28         Hokus.pokus(13, in, out);
```

a) Führen Sie obigen Code (beginnend beim Aufruf in Zeile 28) „gedanklich“ in einem Schreibtischlauf aus. Geben Sie die Werte der Variablen `i`, `ch` und `out` nach *jedem einzelnen* Schleifendurchlauf (also *nach* Zeile 21, aber *vor* 22) tabellarisch in folgender Form an:

i	ch	out
0	...	[..., ..., ...]

Geben Sie bei `out` bitte jeweils den Inhalt des Feldes an (also jeden einzelnen Eintrag).

b) Angenommen die `main`-Methode wird nun wie folgt geändert:

```
char[] in = { 'A', 'u', 'S', ' ', 'i', 's' };
char[] out = { '?', '?', '?', '?', '?', '?' };
Hokus.pokus(13, in, out);
```

Nennen Sie die Zeilennummern in genau der Reihenfolge, in der sie nun nacheinander ausgeführt werden. Geben Sie im Falle der `for(A;B;C)`-Schleife zusätzlich an, welcher Teilausdruck genau ausgeführt wird (also z.B. `3B` für die Prüfung der Bedingung). Schließende Klammern „}“ von Code-Blöcken sollen dabei nicht explizit aufgeführt werden!

Beispiel: $25, 26, 27, 28, 2, 3A, \ldots$

c) Zeichnen Sie den Programmablaufplan (PAP) der Methode `Hokus.pokus`.

Geben Sie Ihre Lösung als `J2AD.pdf` ab.

Gruppenaufgabe 3.3: Gaußsches Eliminationsverfahren — 17 GP

In der Schule haben Sie sicherlich auch das Gaußsche Eliminationsverfahren kennengelernt und wozu man es verwenden kann... In dieser Aufgabe sollen Sie das in Java implementieren.

Laden Sie dazu den Rumpf der Klasse `GaussschesEliminationsverfahren.java` von der AuD-Homepage und ergänzen Sie die drei Methoden gemäß Kommentar im Code.
Das Gaußsche Eliminationsverfahren dient zur Lösung von linearen Gleichungssystemen der Form:

$$\left\{\begin{array}{rcrcrcl} a_{00}\cdot x_0 & + & a_{01}\cdot x_1 & + & a_{02}\cdot x_2 & = & b_0 \\ a_{10}\cdot x_0 & + & a_{11}\cdot x_1 & + & a_{12}\cdot x_2 & = & b_1 \\ a_{20}\cdot x_0 & + & a_{21}\cdot x_1 & + & a_{22}\cdot x_2 & = & b_2 \end{array}\right\} \quad \text{bzw. in Matrix-/Vektor-Schreibweise:} \quad A\vec{x} = \vec{b}$$

In der öffentlichen Testfallmenge wird die Arbeitsweise des Verfahrens, wie Sie es implementieren sollen, schrittweise verdeutlicht, wenn Sie die Testfälle in der angegebenen Reihenfolge abarbeiten:

$$(A|b) := \left(\begin{array}{ccc|c} 3 & 11 & 19 & 101 \\ 5 & 13 & 23 & 103 \\ 7 & 17 & 29 & 107 \end{array}\right) \overset{piv_0}{\longmapsto} \left(\begin{array}{ccc|c} 7 & 17 & 29 & 107 \\ 5 & 13 & 23 & 103 \\ 3 & 11 & 19 & 101 \end{array}\right) \overset{elim_0}{\longmapsto} \left(\begin{array}{ccc|c} 7 & 17 & 29 & 107 \\ 0 & -6 & -16 & -186 \\ 0 & -26 & -46 & -386 \end{array}\right) \overset{piv_1}{\longmapsto}$$

$$\overset{piv_1}{\longmapsto} \left(\begin{array}{ccc|c} 7 & 17 & 29 & 107 \\ 0 & -26 & -46 & -386 \\ 0 & -6 & -16 & -186 \end{array}\right) \overset{elim_1}{\longmapsto} \left(\begin{array}{ccc|c} 7 & 17 & 29 & 107 \\ 0 & -26 & -46 & -386 \\ 0 & 0 & -140 & -2520 \end{array}\right) \overset{loese}{\longmapsto} \vec{x} := \left(\begin{array}{c} -18.0 \\ -17.0 \\ 18.0 \end{array}\right)$$

Sie müssen in dieser Aufgabe keine Fehlerbehandlung implementieren, d.h. Sie dürfen davon ausgehen, dass alle übergebenen Argumente jeweils sinnvoll sind und zueinander passen – insbesondere dass `matrix` und `vektor` ein Gleichungssystem mit einer eindeutigen Lösung beschreiben, so dass beim Aufruf von `loese` garantiert eine diagonalisierte Matrix vorliegt, deren untere/linke Hälfte 0 ist und bei der alle Einträge in der Diagonalen $\neq 0$ sind. Geben Sie Ihre Lösung als `GaussschesEliminationsverfahren.java` über EST ab.

Gruppenaufgabe 3.4: Algebra — 20 GP

In der Schule haben Sie sicherlich auch die Primfaktorzerlegung einer ganzen Zahl $n \geq 1$ kennengelernt und wozu man sie verwenden kann... In dieser Aufgabe sollen Sie das in Java implementieren. Laden Sie dazu den Rumpf der Klasse `Algebra.java` von der AuD-Homepage und ergänzen Sie die drei Methoden wie folgt:

a) Die Methode `primfaktorzerlegung` (sic!) ermittelt die Primfaktorzerlegung ihres Arguments n und gibt sie so als 2-dimensionales Feld z zurück, dass:

$$n = {p_0}^{e_0} \cdot {p_1}^{e_1} \cdot {p_2}^{e_2} \cdot \ldots \cdot {p_{k-1}}^{e_{k-1}}$$

wobei `z[0]` = $\{p_0, p_1, p_2, \ldots, p_{k-1}\}$ die sortierten $(p_0 < p_1 < p_2 < \ldots < p_{k-1})$ Primfaktoren sowie `z[1]` = $\{e_0, e_1, e_2, \ldots, e_{k-1}\}$ ihre jeweiligen Potenzen $(e_i > 0)$ sind.
Wie auch im öffentlichen Testfall festgelegt, ist die Primfaktorzerlegung für $n = 1$ das leere Produkt (also je ein leeres Unterfeld, aber eben *nicht* `null`)!

b) Die Methode `ggT` bekommt die Primfaktorzerlegungen `aPFZ` bzw. `bPFZ` zweier Zahlen `a` und `b` in obiger Feld-Darstellung und soll deren größten gemeinsamen Teiler $ggT(a, b)$ berechnen.

c) Die Methode `kgV` soll das kleinste gemeinsame Vielfache $kgV(a, b)$ zweier Zahlen `a` und `b` berechnen und zurückgeben.

Sie dürfen davon ausgehen, dass alle übergebenen Aktualparameter gültig sind, so dass Sie keine Fehlerbehandlung implementieren müssen. Testen Sie Ihre Lösung gründlich – auch, aber nicht nur mit dem öffentlichen Testfall. Geben Sie Ihre Lösung als `Algebra.java` über EST ab.

23 EP + 37 GP = 60 Punkte