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

2. Übung

Abgabe bis 07.11.2016, 10:00 Uhr

Bitte beachten:

Generelle Hinweise zu den Aufgaben und deren Bearbeitung finden Sie auf dem „organisatorischen Übungsblatt“. Sie können die Aufgaben erst im EST abgeben, nachdem Sie in die Tafelübungen eingeteilt wurden (voraussichtlich Sonntag).

Einzelaufgabe 2.1: Datentypen — 12 EP

Betrachten Sie folgende Fragestellungen und geben Sie zu jeder davon den *bestmöglichen* Java-Datentyp für die darin verwendeten Variablen (jeweils *kursiv gedruckt*, z.B. *plz*) an. Beschränken Sie sich dabei auf *primitive* Typen, `Strings` sowie Reihungen und wählen Sie die Typen so, dass sie möglichst wenig Speicher ver(sch)wenden! Geben Sie bei Reihungen auch die Größe (also die maximale Anzahl an Einträgen) an und *begründen* Sie jeweils kurz Ihre Wahl.

a) Das Programm bekommt eine Postleitzahl plz und prüft, ob es den zugehörigen Ort in Deutschland gibt. Das Ergebnis der Prüfung wird in der Variablen $exists$ abgelegt.

b) Der Taschenrechner soll je zwei kleine Kommazahlen ***ohne*** Genauigkeitsverlust addieren können: Jede Kommazahl z ist positiv sowie kleiner als 1 (also $0 \leq z < 1$) und hat maximal 16 Nachkommastellen. Falls die Summe sum größer als 1 wäre, wird die Vorkommastelle sofort verworfen.

c) In jedem der 14 AuD-Übungsblätter kommen höchstens 15 Teilaufgaben vor. Die nächste EST-Version soll jeder Teilaufgabe eine eindeutige $Kennung$ zuordnen.

d) Ein Physik-Lernprogramm soll die „ungefähre“ Anzahl der Atome N_i folgender Objekte berechnen und auf dem Bildschirm ausgeben.

ein Mensch (70 kg)	:	$N_0 \approx 7,02583 \cdot 10^{27}$ Atome
die Erde	:	$N_1 \approx 6,42390 \cdot 10^{49}$ Atome
die Sonne	:	$N_2 \approx 1,19776 \cdot 10^{57}$ Atome
die Milchstraße	:	$N_3 \approx 2,15596 \cdot 10^{68}$ Atome

Geben Sie Ihre Lösung als `Datentypen.pdf` über EST ab.

Einzelaufgabe 2.2: Logikrätsel — 9 EP

Am 28.10 12:30 wurde die Reihenfolge der F0-F3 sowie A0-A3 geändert, damit es zum Test passt

Logikrätsel lassen sich manchmal mit Hilfe der Aussagenlogik lösen. In folgender Abwandlung eines Rätsels von Raymond Smullyan geht es um Stan, Cartman und Kyle. Einer der drei hat einen Diebstahl begangen und die Ermittlungen haben bis jetzt folgende Fakten ans Tageslicht gezerrt:

F0 Stan sagt, Kyle sei unschuldig.

F1 Cartman behauptet, dass Stan unschuldig sei und

F2 Genau einer und nur einer der drei muss der Dieb sein,

F3 Wer unschuldig ist sagt auf jeden Fall die Wahrheit,

Dieses Rätsel lässt sich wie folgt umschreiben (es gibt nicht unbedingt eine 1-zu-1-Beziehung zwischen F und A:

A0 $\neg\text{stan} \implies \neg\text{kyle}$

A1 $\neg\text{cartman} \implies \neg\text{stan}$

A2 $\text{cartman} \oplus \text{kyle} \oplus \text{stan}$

A3 $(\text{cartman} \implies \neg(\text{kyle} \vee \text{stan})) \wedge$
$(\text{kyle} \implies \neg(\text{cartman} \vee \text{stan})) \wedge$
$(\text{stan} \implies \neg(\text{cartman} \vee \text{kyle}))$

Hierbei bezeichnen die Symbole "$\oplus$" das exklusive Oder und "$a \implies$ b" die Implikation $\neg a \vee b$ ("Falls a, dann auch b"). Wenn einer der drei der Dieb ist, wird die jeweilige Variable auf $true$ gesetzt. Wenn eine Belegung gefunden wird, bei der alle vier Formeln erfüllt werden ($true$ ergeben), ist auch der Dieb gefunden.

a) Laden Sie sich die Datei `Riddle.java` herunter. Diese enthält zunächst nur Gerüst. Vervollständigen Sie die einzelnen Methoden wie unten beschrieben. Ändern Sie *nicht* die Signatur der Methoden.

b) Implementieren Sie als erstes die Methode `implies`, die als Argumente zwei boolsche Werte a und b entgegen nimmt und als Ergebnis $\neg a \vee b$ zurückgibt.

c) Nun soll für die Formeln A0 . . . A3 jeweils eine Methode implementiert werden, die das Ergebnis der jeweiligen Berechnung zurückgibt. Als Parameter erwarten alle diese Methoden eine Belegung der boolschen Variablen für Stan, Cartman und Kyle, in **dieser Reihenfolge**. Die Methoden sollen `a0` bis `a3` heißen.

d) Implementieren Sie nun die Methode `eval`, die überprüft, ob eine bestimmte Belegung alle vier Formeln erfüllt. Die Reihenfolge der Parameter soll gemäß der vorherigen Teilaufgabe **beibehalten** werden.

e) Gemäß Fakt $F0$ muss genau einer der drei der Dieb sein. Um den Dieb zu finden müssen also 3 mögliche Belegungen der Variablen $stan$, $cartman$ und $kyle$ probiert werden.
Implementieren Sie eine Methode `checkRiddle`. Diese Methode soll den Dieb bestimmen, indem alle drei Möglichkeiten probiert werden. Die Methode soll 0 für Stan, 1 für Cartman oder 2 für Kyle zurückgeben. Sollte keine Belegung die Formeln erfüllen (das Rätsel ist dann kaputt), soll -1 zurückgegeben werden.
Hinweis: Ihre Implementierung soll auch für andere Rätsel (also geänderte `a0` bis `a3`) funktionieren. Es reicht hier folglich nicht, den Dieb von Hand zu bestimmen und einzutragen.

Geben Sie Ihre Lösung als `Riddle.java` im EST ab.

Gruppenaufgabe 2.3: Datentypen und Ausdrücke — 14 GP

Gegeben sei folgender Code-Abschnitt:

```
int x = 20, y = 2;
double z = 0.815;
```

Werten Sie jeden der folgenden Ausdrücke *einzeln und jeweils unmittelbar* nach dem obigen Code aus, indem Sie zuerst den Wert der `rechten Seite` ermitteln. Geben Sie zusätzlich an, welchen Datentyp jeweils die Variable besitzen muss, welcher der Wert des Ausdrucks zugewiesen wird.

a) `a = (y > 1) && (5 != 4) || (x > -4);`

b) `b = +x++ + ++x - -x-- - --x;`

c)
```
c = -2;
c *= z;
```

d) `d = x | 5;`

e) `e = ((x == y) || (x < z * y));`

f) `f = y << y;`

g) `g = !((x >> 666 < y) & (z++ == x));`

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

Gruppenaufgabe 2.4: Variablen, Datentypen und Arrays — 16 GP

Laden Sie den vorgegeben Rumpf des Programms Variablen.java von der AuD-Homepage und ergänzen Sie die mit `// TODO:` markierten Stellen gemäß der dort angegebenen Kommentare. Testen Sie Ihre Änderungen, indem Sie das Programm übersetzen und ausführen – gerne können Sie in der `main`-Methode weitere Tests hinzufügen.
Geben Sie Ihre fertig ergänzte Datei `Variablen.java` über EST ab.

Gruppenaufgabe 2.5: Arrays verwenden - Praxis — 9 GP

Sie spielen ein einfaches Zahlenspiel gegen einen Freund:
Jeder von Ihnen würfelt zweimal. Die geworfenen Ziffern werden notiert - zunächst die Punkte des Spielers 1, dann die des zweiten Spielers (siehe Ausführungsbeispiele). Nun werden die beiden Augensummen berechnet. Wer die höhere Augensumme erwürfelt, hat gewonnen.
Sie finden dazu auf der Übungshomepage eine Datei `GameOfDice.java`. Vervollständigen Sie die Methode `game`. Die Methode soll je nach Gewinner 1 oder 2 zurückgeben. Bei einem Unentschieden 0 und falls das Spiel fehlerhaft ist (falsche Anzahl an Zahlen) -1. Sie können davon ausgehen, dass alle Zahlen gültige Würfelergebnisse sind.
Hinweis: Suchen Sie bei Bedarf nach der Dokumentation zu `Integer.parseInt`.

Ausführungsbeispiele:

- ```
  > java GameOfDice 4 6 1 3
  Player 1 won
  ```
- ```
  > java GameOfDice 2 6 5 4
  Player 2 won
  ```
- ```
  > java GameOfDice 3 5 4 4
  Draw
  ```

21 EP + 39 GP = 60 Punkte
  ```