Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion

Priv.-Doz. Dr. Frank Huch, Christoph Daniel Schulze, Sandra Dylus

Institut für Informatik

6. Klausur zur Vorlesung „Informatik für Nebenfächler“

WS 14/15

Hinweise zur Klausur:

- Sie dürfen jegliches Material, allerdings keine mitgebrachten elektronischen Geräte verwenden. Mobiltelefone bitte ausschalten.
- Zur Bearbeitung der Aufgaben sollen Sie Ihren iLearn-Account verwenden, allerdings nicht mit Ihrem normalen Passwort, sondern dem iLearn-Passwort auf Ihrem Klausurdeckblatt.
- Bei technischen Problemen kontaktieren Sie uns bitte frühzeitig. Gegebenenfalls werden wir zum Ausgleich technischer Probleme die Bearbeitungszeit verlängern, also keine Panik.
- Die Einlesezeit beträgt 10 Minuten. Diese Zeit soll ausschließlich dazu dienen, die Klausuraufgaben zu lesen. Sie dürfen in dieser Zeit nicht schreiben.
- Während der Einlesezeit werden wir Ihre Identität anhand Ihres Lichtbild- und Ihres Studienausweises überprüfen. Bitte legen Sie diese gut sichtbar auf Ihrem Tisch bereit. Unterschreiben Sie das Deckblatt erst, wenn wir vorbeikommen.
- Die anschließende Bearbeitungszeit beträgt 150 Minuten. In den letzten 20 Minuten bitten wir Sie, zur Vermeidung von Hektik und Unruhe nicht mehr vorzeitig abzugeben.
- Bitte notieren Sie auf jedem Blatt, das Sie abgeben möchten, Ihren vollständigen Namen und Ihre Matrikelnummer. Bei Abgaben im iLearn ist dies nicht erforderlich.
- Am Ende der Bearbeitungszeit werden wir die Klausur an Ihrem Platz zusammenheften und einsammeln. Ihre elektronischen Abgaben werden wir zur Korrektur ausdrucken.
- Die Bekanntgabe der Noten und die Einsicht in die Korrekturen findet am 17. April zwischen 10 und 11 Uhr in Raum 715 im Hochhaus statt.
- Sie benötigen 35 Punkte um die Klausur zu bestehen. Die maximal erreichbare Punktzahl beträgt 80 Punkte. Sie können also eine 1,0 erreichen, auch wenn Sie nur 70 Punkte bearbeiten. Werden alle Aufgaben bearbeitet, so können fehlende Punkte aus anderen Aufgaben ausgeglichen werden.
- Teilen Sie sich Ihre Zeit gut ein. Beißen Sie sich nicht an (noch) fehlerhaften Programmen fest.

Aufgabe 1 - Arrayausdrücke

12 Punkte

(a) Für Ausdrücke, die aus Additionen und aus Multiplikationen bestehen, ist folgende EBNF gegeben:

```
E    ::= E '+' E1 | E1
E1   ::= E1 '*' E2 | E2
E2   ::= '(' E ')' | Val
Val  ::= 'x' | 'y' | 'z' | '2' | '4' | '42'
```

Prüfen Sie, ob folgende Ausdrücke in dieser EBNF ableitbar sind indem Sie, falls möglich, einen Ableitungsbaum angeben oder, sonst, begründen, warum dies nicht möglich ist.

1. `x + y + z`
2. `x * 4 + 2`
3. `((x) * y) + z`

(b) Ein Array über Zahlen in Ruby ist entweder ein leeres Array oder ein Array, in dem jedes Element entweder eine Zahl oder ein weiteres Array über Zahlen ist. Beispiel:

```
[42, 73, [42, [73]]]
```

Definieren Sie eine EBNF für Arrays über Zahlen in Ruby. Die Zahlen selbst brauchen Sie dabei in ihrer Grammatik nicht zu definieren, sondern können auf folgende EBNF-Regeln zurückgreifen:

```
Num ::= ['-'] Dig {Dig}
Dig ::= '0' | ... | '9'
```

Aufgabe 2 - Programmpunkttabelle

10 Punkte

Wir betrachten folgendes Ruby-Programm:

```
n = 2;                  #1
m = 3;                  #2
r = 1;                  #3

for i in 1 .. m do      #4
  x = 0;                #5

  for j in 1 .. n do    #6
    x = x + r;          #7
  end;                  #8

  r = x;                #9
end;                    #10

puts(r);                #11
```

(a) Simulieren Sie das Programm mithilfe einer Programmpunkttabelle.

(b) Was berechnet das Programm abhängig von einer initialen Belegung von `n` und `m` mit ganzen Zahlen größer Null?

Aufgabe 3 - Tabellensumme

12 Punkte

(a) Schreiben Sie eine nicht mutierende Funktion `array_sum(a)`, welche die Summe der Zahlen in einem eindimensionalen Array `a` zurückliefert.

(b) Schreiben Sie eine nicht mutierende Funktion `row_sum(a)`, welche die Funktion `array_sum` verwendet, um zu einem gegebenen zweidimensionalen Array `a` ein neues, eindimensionales Array zurückzuliefern, dessen Einträge den jeweiligen Zeilensummen in `a` entsprechen.

Beispiel

```
array = [[1, 2, 3], [2, 4, 6], [4, 8, 12]]
p(row_sum(array))    # Ausgabe: [6, 12, 24]
```

(c) Schreiben Sie eine mutierende Prozedur `row_sum!(a)` ähnlich wie die Funktion `row_sum(a)`, welche allerdings direkt die Zeilen in `a` durch deren Zeilensumme ersetzt statt ein neues Array zurückzuliefern.

Aufgabe 4 - Aufzählen und Überprüfen

10 Punkte

Eine Zahl heißt Hammingzahl, wenn Sie sich allein aus den (gegebenfalls mehrfach vorkommenden) Primfaktoren 2, 3 und 5 zusammensetzt. Jede Hammingzahl kann also dargestellt werden als $2^i * 3^j * 5^k$.

In diese Aufgabe sollen Sie mittels zweier unterschiedlicher Ansätze ein Funktion `is_hamming` definieren, welche für eine gegebene Zahl überprüft, ob es sich um eine Hammingzahl handelt. Das Ergebnis soll ein boolescher Wert sein, welcher anzeigt, ob es sich um eine Hammingzahl handelt oder nicht.

(a) Implementieren Sie die Funktion `is_hamming(n)`, die unter Verwendung des Programmierschemas *Aufzählen und Überprüfen* testet, ob die Zahl `n` eine Hammingzahl ist. In dem Fall soll sie `true` zurückgeben, ansonsten `false`. Zählen Sie hierzu die möglichen Werte für i, j und k bis hin zur Wurzel von `n` auf und überprüfen Sie dann, ob sich mit obiger Formel die zu überprüfende Zahl ergibt.

(b) Das Verfahren in (a) ist recht ineffizient. Geschickter ist es, die Zahl nacheinander durch die Primfaktoren 2, 3 und 5 zu teilen, bis dies nicht mehr möglich ist. Ergibt sich dann eine Zahl > 1, so handelte es sich um keine Hammingzahl. Ergibt sich die Zahl 1, so war die Zahl eine Hammingzahl. Implementieren Sie eine Funktion `is_hamming_efficient(n)`, welche dieses effiziente Verfahren implementiert. Für das überprüfen, ob eine Zahl durch eine andere teilbar ist, kann man mit Hilfe des Operators % den Rest der ganzzahligen Division bestimmen, welcher bei Teilbarkeit 0 ergibt.

Im Beispiel ergibt sich für die Zahl 20:

```
20%2 == 0 , d.h. 20 ist durch 2 teilbar, mache weiter mit 20/2 = 10
10%2 == 0 , d.h. 10 ist durch 2 teilbar, mache weiter mit 10/2 = 5
5%2  == 1 , d.h. 5 ist nicht durch 2 teilbar
5%3  == 2 , d.h. 5 ist nicht durch 3 teilbar
5%5  == 0 , d.h. 5 ist durch 5 teilbar, mache weiter mit 5/5 = 1
1%5  == 1 , d.h. 1 ist nicht durch 5 teilbar

Die letzte Zahl lautete 1, d.h. es gibt keine weiteren Primfaktoren
und die Zahl ist eine Hammingzahl.
```

Aufgabe 5 - Methodenschreibweise

12 Punkte

Geben Sie zu den folgenden Aussagen an, ob sie wahr oder falsch sind. Begründen Sie ihre Antwort und geben Sie, wenn möglich, hierzu auch geeignete Belegungen der vorkommenden Variablen an.

1. `a + b` kann ein Ausdruck sein, der ein String-Objekt als Ergebnis hat.
2. `a + b` kann ein Ausdruck sein, der eine Zahl als Ergebnis hat.
3. `a + b` kann ein Ausdruck sein, der einen booleschen Wert als Ergebnis hat.
4. `a * b` hat nur ein Ergebnis, wenn sowohl `a` als auch `b` eine Zahl ist.
5. `c[2]` ist eine Anweisung.
6. `c[2] = 42` ist keine Zuweisung, aber ein Ausdruck.
7. `a[1][3] = 73` besteht aus zwei Methodenaufrufen, von denen einer nicht mutierend und einer mutierend ist.
8. `b[1][3]` kann den Wert 42 als Ergebnis haben.
9. Der Name nicht mutierender Funktionen, Prozeduren bzw. Methoden sollte in Ruby immer ein Ausrufezeichen enhalten, wie zum Beispiel die String-Methode `reverse!`.
10. Die Ausdrücke `a ** b ** c` und `a ** (b ** c)` sind gleich.

Aufgabe 6 - Reguläre Ausdrücke 12 Punkte

(a) Geben Sie einen regulären Ausdruck an, der Datumsangaben im deutschen Format 27.03.2015 matcht. Dabei sollen folgende Annahmen gelten:

- Für Tag und Monat sollen Zahlen mit nur einer Ziffer erlaubt sein.
- Tage dürfen nur zwischen 1 und 31 liegen.
- Monate dürfen nur zwischen 1 und 12 liegen.
- Jahre müssen mindestens einstellig sein und dürfen nicht mit einer Null beginnen falls sie aus mehr als einer Ziffer bestehen.

(b) Benutzen Sie Ihren regulären Ausdruck aus Aufgabenteil (a) um ein Ruby-Programm zu schreiben, welches in einem gegebenen Text alle Datumsangaben im deutschen Format ins amerikanische Format (03/27/2015) überführt und den resultierenden Text auf der Konsole ausgibt. Der gegebene Text soll vom Benutzer über die Konsole abgefragt werden.

Beispiel:

```
Die Klausur für 5 ECTS-Studenten findet am 27.03.2015 statt, die für alle anderen
am 30.03.15. Die Klausureinsicht findet am 06.07.15 statt.
```

Ausgabe:

```
Die Klausur für 5 ECTS-Studenten findet am 03/27/2015 statt, die für alle anderen
am 03/30/15. Die Klausureinsicht findet am 07/06/15 statt.
```

Aufgabe 7 - Von Drachen und Helden 12 Punkte

Jeder mag Drachen und Helden! Stellen Sie sich vor, der König hätte Sie beauftragt, eine Datenbank zur Verwaltung von Drachen und Helden zu basteln. Sie sollen dazu folgende Sachverhalte abbilden:

- Ein Drache hat einen Namen (z.B. “Pikus”) und einen Zusatz (z.B. “Der Geschuppte”), sowie ein Vermögen an Goldstücken.
- Ein Held hat einen Vornamen (z.B. “Herald”), einen Nachnamen (z.B. “Brotfleisch”), sowie selbstredend auch einen Zusatz (z.B. “Der Niederstrecker von Generationen”).
- Im Königreich gibt es Städte, die jeweils einen Namen tragen (z.B. “Bruchstadt”).
- Jede Stadt hat genau einen Ehrenhelden, welcher den die Stadt bedrohenden Drachen getötet und ihr im Gegenzug im Tausch für eine Maid dessen Vermögen überreicht hat (ja, jede Stadt wird nur von einem einzigen Drachen bedroht).
- Jeder Drachen kann von mehreren Helden bekämpft worden sein, und jeder Held wird im Laufe seines (erwartungsgemäß kurzen) Lebens zumindest potentiell mehrere Drachen bekämpfen.

(a) Zeichnen Sie das ER-Modell mit den benötigten Tabellen, deren Attributen und deren Beziehungen.

(b) Eliminieren Sie gegebenfalls auftretende n:m-Beziehungen. Falls Sie dies schon in Aufgabenteil (a) erledigt haben geben Sie hier bitte an, welche n:m-Beziehungen es gegeben hätte und wie Sie diese aufgelöst haben.

(c) Geben Sie konkrete Tabellen an, bei denen die gewählten Beziehungen auch genutzt werden.

(d) Geben Sie eine SQL-Abfrage für Ihre Datenbank an, welche die Namen aller Städte zusammen mit den Namen, Vornamen und Zusätzen ihrer jeweiligen Helden auflistet. Sie brauchen hier kein vollständiges Ruby-Programm zu schreiben, sondern nur die eigentliche SQL-Abfrage angeben.