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

2. 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.
- Bitte notieren Sie auf jedem Blatt, das Sie abgeben möchten, Ihren vollständigen Namen und Ihre Matrikelnummer. Bei Abgaben im iLearn ist dies nicht erforderlich.
- Da die Klausur im Anschluss direkt noch einmal geschrieben wird, können Sie leider nicht früher abgeben und gehen. Sollte dies Probleme ergeben, kontaktieren Sie uns bitte frühzeitig.
- 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 20. Februar 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 - Objektidentität 12 Punkte

(a) Die `*`-Methode der String-Klasse ist nicht mutierend. Vielmehr wird ein neues Objekt konstruiert. Erläutern Sie mit Hilfe des folgenden Programmabschnitts, weshalb es selbst für den Fall, dass der String nur mit 1 multipliziert wird, wichtig ist, dass ein neues Objekt angelegt wird.

```
x = "a";       #1
y = x*i;       #2
y[0,1] = "b"; #3
```

Überlegen Sie hierzu, was bei unterschiedlichen Werten für `i` passiert und wie die Variablen `x` und `y` am Ende dieses Codestücks belegt sind.

(b) Implementieren Sie eine zweistellige, nicht mutierende Funktion `without(s, n)`, welche einen String `s` sowie eine ganze Zahl `n` erwartet. Die Funktion soll einen String zurückliefern, der `s` entspricht, wo allerdings jedes `n`-te Zeichen entfernt wurde. Beispiele:

```
str = "Haarfestiger";
puts(without(str, 1)) -> ""
puts(without(str, 2)) -> "Hafsie";
puts(without(str, 3)) -> "Harfstge";
```

Ist `n < 1` oder `n >= str.length` wird kein Zeichen entfernt. Achten Sie darauf, dass Ihre Funktion die Konvention für nicht mutierende Funktionen aus der Vorlesung erfüllt.

(c) Implementieren Sie eine mutierende Variante der `without`-Funktion, so dass die Prozedur `without!(s, n)` im String `s` alle `n`-ten Zeichen mutierend entfernt.

Aufgabe 2 - Comicdatenbank 12 Punkte

In dieser Aufgabe sollen Sie eine Datenbank zur Verwaltung von Comics erstellen. Sie sollen dazu folgende Sachverhalte abbilden:

- Eine Comicserie trägt einen Titel sowie die Adresse einer zugehörigen Webseite.
- Zu einer Comicserie gibt es konkrete Comics. Jeder Comic hat eine laufende Nummer, die nur innerhalb der Comicserie eindeutig ist. Darüber hinaus hat jeder Comic einen Titel, einen Autoren, sowie einen Zeichner.
- Autoren und Zeichner haben einen Vornamen und einen Nachnamen.
- Comics werden in Comicheften veröffentlicht. Jedes Comicheft besteht aus beliebig vielen Comics und hat ein Veröffentlichungsdatum. Jeder Comic kann in beliebig vielen Comicheften veröffentlicht werden.

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

(b) Eliminieren Sie gegebenfalls auftretende n:m-Beziehungen.

(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 laufende Nummer und die Titel aller Comics der Comicserie “Asterix” aufsteigend nach laufender Nummer sortiert zurückliefert. Sie brauchen hier kein vollständiges Ruby-Programm zu schreiben, sondern nur die eigentliche SQL-Abfrage angeben.

Aufgabe 3 - Programmpunkttabelle

10 Punkte

Betrachten Sie folgendes Ruby-Programm:

```
n = 1337                          #1

r = 0                             #2
z = 10**(n.to_s().size - 1)       #3
while z > 0 do
  s = n / z                       #4
  r = r + s                       #5
  n = n - s * z                   #6
  z = z / 10                      #7
end                               #8

puts(r.to_s())                    #9
```

(a) Simulieren Sie das Programm mithilfe einer Programmpunkttabelle. Beachten Sie dabei, dass die Methode `/` für ganze Zahlen verwendet wird.

(b) Was berechnet das Programm abhängig von einer initialen Belegung von `n` (einer gegebenen ganzen Zahl größer Null)?

Aufgabe 4 - Aufzählen und Überprüfen

10 Punkte

Ein *pythagoreisches Tripel* besteht aus drei positiven Zahlen a, b und c für die gilt:

$$a^2 + b^2 = c^2$$

Zum Beispiel bilden die Zahlen $a = 3$, $b = 4$ und $c = 5$ ein pythagoreisches Tripel.

(a) Schreiben Sie ein Ruby-Programm, welches alle pythagoreischen Tripel auf der Konsole ausgibt, deren beteiligte Zahlen zwischen einschließlich 1 und 100 liegen. Benutzen Sie dazu das Programmierschema *Aufzählen und Überprüfen*.

(b) Optimieren Sie ihr Programm, so dass nur zwei der drei Variablen geraten werden. Hierzu können Sie Zahlen des Typs Float mit Hilfe der Methode `to_i` in eine ganze Zahl umwandeln. Außerdem könnte in diesem Kontext die Verwendung der Funktion `Math.sqrt` hilfreich sein.

Aufgabe 5 - Programmieren mit Arrays

12 Punkte

In dieser Aufgabe sollen Sie Funktionen beziehungsweise Prozeduren definieren, welche in einem gegeben Array alle vorkommenden Elemente verdoppeln. Das Verhalten sollte anhand des folgenden Beispielaufrufs klar werden:

```
double([7,42,73]) -> [7,7,42,42,73,73]
```

(a) Definieren Sie eine nicht mutierende Funktion `double(a)`, welche obiges Verhalten zeigt.

(b) Definieren Sie eine mutierende Prozedur `double!(a)`, welche ein übergebenes Array-Objekt entsprechend mutiert.

(c) Definieren Sie eine nicht mutierende Funktion `to_table(a, n)`, welche ein Array `a` in ein zweidimensionales Array umwandelt. Jede Zeile soll dabei das entsprechende Element `n`-mal enthalten.

Beispiel:

```
to_table([7,42,73], 3)   # --> [[7,7,7], [42,42,42], [73,73,73]]
```

Aufgabe 6 - Einfaches HTML 12 Punkte

Die *Hypertext Markup Language* (HTML) wird zur Definition von Webseiten verwendet. Die folgende EBNF beschreibt einen vereinfachten Teil der Sprache HTML:

```
HTML ::= '<html>' Cont '</html>'

Cont ::= Cont Cont
       | '<i>'  Cont '</i>'
       | '<p>'  Cont '</p>'
       | '<br>' Cont
       | Text

Text ::= {Sym}

Sym  ::= 'a' | ... | 'z' | 'A' | ... | 'Z'
```

Struktur- und Formatierungsinformationen der Dokumente werden mithilfe sogenannter *Tags* vorgenommen. Hierbei handelt es sich um Zeichenfolgen innerhalb der spitzen Klammern, also die Tags `<html>` (für das gesamte HTML-Dokument), `<i>` (kursive Schrift, englisch *italic*), `<p>` (zur Definition eines Absatzes) und `<br>` (für einen Zeilenumbruch). Bis auf das Tag `<br>` werden alle Tags durch ein passendes schließendes Tag (mit einen Schrägstrich vor dem Tag-Namen) geschlossen.

(a) Geben Sie für die folgenden HTML-Dokumente Ableitungsbäume an oder begründen Sie, warum diese nicht abgeleitet werden können.

- `<html>Hi<br></html>`
- `<html><i><p>Hi</i></p></html>`
- `<html><i>H</i>e<i>y</i></html>`

(b) Wir betrachten folgende alternative Definition für HTML-Dokumente:

```
HTML2   ::= '<html>' Cont2 '</html>'

Cont2   ::= Cont2 Cont2
        | '<' TagName '>' Cont2 '</' TagName '>'
        | '<' TagName '>' Cont2
        | Text2

TagName ::= 'i' | 'p' | 'br'

Text2   ::= {Sym2}

Sym2    ::= 'a' | ... | 'z' | 'A' | ... | 'Z'
```

Wird von dieser EBNF über das Nichtterminalsymbol `HTML2` dieselbe Sprache definiert wie durch das Nichtterminalsymbol `HTML` in der ersten EBNF? Begründen Sie Ihre Aussage.

(c) HTML haben wir mit Hilfe einer EBNF definiert. Insbesondere in der Definition des Nichtterminalsymbols `Text` verwenden wir die EBNF-Konstrukte {} und eine Alternative innerhalb einer Gruppierung. Definieren Sie eine BNF, welche genau dieselbe Sprache beschreibt, wie die gegebene EBNF.

Aufgabe 7 - Breitengrade 12 Punkte

In dieser Aufgabe sollen Sie Breitengrade aus einer Zeichenkette einlesen und anders formatiert wieder ausgeben.

Ein Breitengrad besteht aus den folgenden Komponenten:

1. Eine Gradzahl (zwischen 0 und 90) gefolgt von dem Gradsymbol (`°`)
2. Eine Minutenzahl (zwischen 0 und 59) gefolgt von einem Minutenstrich (`'`)
3. Eine Sekundenzahl (zwischen 0 und 59) gefolgt von zwei Sekundenstrichen (`''`)
4. Eine Nord-/Südangabe (N oder S)

Zusätzlich gilt, dass `90°0'0''N` beziehungsweise `90°0'0''S` die maximalen Breitengrade sind. Außerdem wollen wir Zahlen mit führender Null verbieten.

Gültige Breitengrade sind zum Beispiel `54°20'0''N` (Breitengrad von Kiel), `90°0'0''S` (Südpol), `0°0'0''N` (Äquator) und `54°20'19''N` (Ihr aktueller Sitzort). Keine gültigen Breitengrade sind `90°5'0''S` (über 90°), `42°21'10''` (keine Nord-/Südangabe) und `1°02'3''N` (führende Null bei `02'`).

(a) Definieren Sie einen regulären Ausdruck, welcher genau Breitengrade in dieser Form matcht.

(b) Schreiben Sie ein Programm, welches einen Breitengrad in der oben definierten Formatierung vom Benutzer einliest und als Dezimalgrad wieder ausgibt. Der Dezimalgrad (Dezimalzahl zwischen -90.0 und 90.0) kann anhand der folgenden Formel berechnet werden:

$$Dezimalgrad = Gradzahl + \frac{Minutenzahl}{60} + \frac{Sekundenzahl}{3600}$$

Zusätzlich werden südliche Breitengrade mit -1 multipliziert. Achten Sie darauf, dass es sich beim Dezimalgrad nicht um eine ganze Zahl handelt.

Falls die Eingabe kein korrekter Breitengrad ist, soll der Benutzer erneut zur Eingabe aufgefordert werden, bis ein gültiger Breitengrad eingegeben wurde.