Programmierparadigmen

Sommerersemester 2017
Martin Wittiger, Felix Krause, Timm Felden

2. Übung

Abgabe bis 3. Mai um 4:44
Beachten Sie die Abgabehinweise auf der Vorlesungswebseite!

Aufgabe 2.1 ..... 2 Punkte

Dateiname: `eins.pdf` ★☆☆

1. Nennen Sie fünf Ausführungsmodelle für Programmiersprachen und geben Sie zu jedem dieser Ausführungsmodelle eine charakteristische Eigenschaft an, die es von allen anderen unterscheidet.

2. Nennen Sie fünf Eigenschaften, die bei Programmiersprachen wünschenswert sind.

3. Nennen Sie fünf Eigenschaften, die aus Sicht eines Softwareentwicklers bei einem Programm wünschenswert sind.

Aufgabe 2.2 ..... 6 Punkte

Dateiname: `Loesung.java` ★★★

Hannah hat von einer Freundin einen Stein erhalten. Die Freundin behauptet, dass der Stein genau z Gramm wiegt. Hannah hat eine Balkenwaage und will die Behauptung damit überprüfen. Sie will dazu auf die eine Seite der Waage den Stein legen und auf die andere Seite einige Gewichte, die in Summe genau z Gramm schwer sind.

Hannah stehen n Schachteln $S_1 \dots S_n$ mit Gewichten zur Verfügung. Dabei enthält die Schachtel S_i genau q_i gleiche Gewichte, die jeweils 10^{g_i} Gramm wiegen. Hannah will so wenige Schachteln wie möglich öffnen.

In dieser Aufgabe sollen Sie eine Methode vervollständigen, die Hannah dabei hilft die richtigen Schachteln auszuwählen. Die Methode erhält ein Zielgewicht und eine Liste von Schachteln als Parameter. Sie gibt dann eine Liste von Schachteln zurück, sodass mit den in diesen Schachteln enthaltenen Gewichten das Zielgewicht zusammengestellt werden kann. Dabei ist sichergestellt, dass die Liste von minimal möglicher Länge ist.

Angenommen Hannah hat 4 Schachteln mit jeweils 10 Gewichten, wobei die Gewichte 1 Gramm, 10 Gramm, 100 Gramm und 1000 Gramm schwer sind. Ist das Zielgewicht nun 1001 Gramm muss Hannah zwei Schachteln öffnen, nämlich die 1-Gramm-Schachtel und entweder die 100-Gramm-Schachtel oder die 1000-Gramm-Schachtel. Auch wenn das Zielgewicht 11 000 Gramm beträgt reichen zwei Schachteln aus. Ein Zielgewicht von 15 000 Gramm lässt sich mit den Schachteln gar nicht erreichen.

Auf der Webseite der Vorlesung finden Sie ein Gerüst für ein Java-Programm. In diesem sollen Sie die Funktion `loese` implementieren. Ändern Sie dabei nur die Datei `Loesung.java`. Sie können das Main-Programm und die Testdateien `a.txt` – `e.txt` dazu nutzen Ihr Programm zu testen.

Überlegen Sie sich weitere Tests und prüfen sie Ihre Abgabe damit. Damit das Problem nicht zu schwer wird, hier noch ein paar Einschränkungen bezüglich der verwendeteten Zahlen:

- n, z, alle g_i und alle q_i sind ganze Zahlen
- $0 \leq n \leq 8\,000$
- $0 \leq z \leq 10^{18}$
- $\forall i \in \{1 \dots n\}\ 0 \leq q_i \cdot 10^{g_i} \leq 10^{18}$

Wenn Sie feststellen, dass das Gesamtgewicht nicht mit den gegebenen Schachteln erreicht werden kann, geben Sie `null` zurück.

Aufgabe 2.3 ..... 4 Punkte

Dateiname: `suv.hs`

Ein Diesel-SUV hat eine Reichweite von r km. Das heißt er kann mit vollem Tank r km fahren, bevor er wieder aufgetankt werden muss. Peter will eine Strecke aus n Etappen fahren. Am Ende jeder Etappe kann er entweder weiterfahren oder seinen SUV wieder voll auftanken.

Peter möchte wissen, ob er eine Gesamtstrecke bestehend aus einigen Etappen mit einer initialen Tankfüllung t mit maximal s Tankstopps bewältigen kann. Sie sollen dazu eine Haskell-Funktion `ok` schreiben. Dabei soll »`ok s r t es`«»`True`«ergeben, wenn Peter es schafft, die Etappen `es` (in der gegebenen Reihenfolge) mit seinem SUV mit Reichweite `r` und einer initialen Tankfüllung `t` mit maximal `s` Tankstopps zu fahren. Ist es nicht zu schaffen, gibt die Funktion `False` zurück.

Beispiele:

- `ok 2 10 10 [7, 3, 5, 4, 4] == True`
  Peter kann nach der zweiten und nach der dritten Etappe einen Tankstopp einlegen.
- `ok 2 10 10 [7, 3, 5, 6, 7] == False`
  Die Gesamtstrecke ist nicht mit zwei Stopps zu bewältigen.
- `ok 5 6 6 [1, 2, 6, 7, 2] == False`
  Die Etappe der Länge 7 ist nicht zu schaffen.

Fügen Sie Ihre Funktion in das zur Verfügung gestellte Script ein. Sie können entweder über das rudimentäre Kommandozeileninterface oder mit GHCi testen. Etappenlängen sind nicht-negativ. Sie können davon ausgehen, dass die aktuelle Tankfüllung nie größer als die Reichweite ist.

Aufgabe 2.4 ..... 4 Punkte

Dateiname: `refcount.pdf` ★★☆

In dieser Aufgabe geht es um Reference-Counting. Wir werden Reference-Counting einer implementierung von Doubly-Linked Lists mit Dummy Nodes besprechen. Machen Sie sich zunächst mit dem Listenimplemnetierungskonzept »Dummy Nodes« vertraut, wenn Sie sich damit nicht ohnehin schon auskennen.

Auf der Webseite der Vorlesung finden Sie ein Ada-Package. In diesem wird eine Liste implementiert. Die meisten öffentlich sichtbaren Funktionen haben wir ausgelassen, wir wollen hier vor allen die private Funktion `Remove_Between` untersuchen. Bei der Implementierung wird davon ausgegangen, dass Reference-Counting verwendet wird.

Beschreiben Sie (mit Bezugnahme auf Reference Counting), was im Speicher beim Aufruf von `Remove_Between` passiert wenn, ...

1. ... kein Knoten zwischen From und To liegt (`From.Forward = To`).
2. ... ein Knoten zwischen From und To liegt (`From.Forward.Forward = To`).
3. ... mehr als ein Knoten zwischen From und To liegt.

Ihre Beschreibung sollte die genaue Zahl an Referenzen, die es auf die relevanten Zellen gibt, beinhalten. Erklären Sie, in welchen Fällen es zu einem Memory Leak kommt.

Erklären Sie, wie man die auftretenden Memory Leaks durch Hinzufügen von einigen Zeilen Code vermeiden kann. Geben Sie tatsächlichen Ada-Code an.

Aufgabe 2.5 ..... 2 Punkte

Dateiname: `paramval.pdf` ★☆☆

Gegeben sei das folgende Programm in einer Ada-ähnlichen Sprache, die es im Gegensatz zu Ada ermöglicht, einen Parameterübergabemechanismus durch einen Compiler-Schalter auszuwählen, so dass dann alle Parameter mit diesem Mechanismus übergeben werden. Auf alle formalen Parameter seien auch schreibende Zugriffe zulässig.

```
1  procedure Params is
2     X : Integer := 2;
3     Y : Integer := 5;
4     Z : Integer := 1;
5
6     A : array (1 .. 5) of Integer := (2, 3, 1, 4, 5);
7
8     procedure G (X : in out Integer; Y : in out Integer) is begin
9        A (Y) := A (A (Y));
10       Z := X;
11       Y := Y + X;
12    end G;
13
14    procedure F (X : in out Integer; Y : in out Integer) is
```

```
15          Z : Integer := 2;
16       begin
17          X := A (Z);
18          Y := A (X);
19          G (X, Z);
20          A (X) := A (Z);
21       end F;
22
23    begin
24       F (Z, A (Y));
25
26       Put (X); Put (Z);
27       Put (A (1)); Put (A (2)); Put (A (3)); Put (A (4)); Put (A (5));
28    end Params;
```

Geben Sie für die beiden folgenden Übergabemechanismen an, welche Ausgaben das Programm produziert, und erklären Sie jeweils, wie diese Ausgaben zustande kommen. Sollte das Programm auf das Array mit falschem Index zugreifen, geben Sie den Zustand der globalen Variablen zu diesem Zeitpunkt an. Gehen Sie dabei davon aus, dass jegliche Optimierungen des Compilers deaktiviert sind.

1. call-by-value
2. call-by-value/result