Programmierparadigmen

Sommerersemester 2017
Martin Wittiger, Felix Krause, Timm Felden

1. Übung

Abgabe bis 24. April um 4:44
Beachten Sie die Abgabehinweise auf der Vorlesungswebseite!

Vorwort

Dieses Übungsblatt dient dazu, Sie mit dem Abgabesystem und den Richtlinien vertraut zu machen. Daher sind die Aufgaben einfach gehalten und geben wenig Punkte. Da Sie mit hoher Wahrscheinlichkeit auf Ihnen unbekannte Programmiersprachen treffen werden, werden Sie ein wenig recherchieren müssen, um dieses Blatt zu bearbeiten. Wir legen Ihnen nahe, diesen Aufwand auch tatsächlich zu betreiben, denn sonst müssen Sie ihn bei späteren Aufgaben nachholen.

Aufgabenstellung

Nehmen Sie an, Sie seien ein Übungsgruppenleiter. Ihre Studenten müssen sechs Aufgabenblätter bearbeiten, jedes Aufgabenblatt gibt 10 Punkte. Um Ihnen die Scheinvergabe zu erleichtern, möchten Sie sich ein kleines Programm schreiben, in welches Sie die erreichten Punkte eines Studenten für jedes der sechs Übungsblätter eingeben. Das Programm soll dann zum Einen die durchschnittlichen Punkte pro Blatt ausgeben und zum Anderen überprüfen, ob der Student mindestens 50% der Punkte erreicht hat.

Die sechs Punktzahlen werden dem Programm als Kommandozeilenparameter übergeben. Die Ausgabe beinhaltet zwei Zeilen, die jeweils durch einen Zeilenumbruch abgeschlossen werden. In der ersten Zeile soll die Durchschnittspunktzahl stehen; in der zweiten Zeile „`Ja`“, wenn der Student mindestens 50% der Punkte erreicht hat, „`Nein`“ sonst. Beispiele (die Ausgabe ist fett hervorgehoben):

```
$ ./scheintool 3 7 10 4 6 6
6
Ja
$ ./scheintool 3 3 7 2 5 4
4
Nein
```

Da Sie bekannterweise unfehlbar sind, können Sie davon ausgehen, dass Sie immer die korrekte Anzahl an Zahlen in das Programm eingeben. Einer Weissagung der Schicksalsfrau Urd folgend können Sie weiterhin davon ausgehen, dass der Durchschnitt immer eine Ganzzahl sein wird.

Bearbeitung

Nachfolgend finden Sie Programmcode in verschiedenen Programmiersprachen, welcher Kommandozeilenparameter als Ganzzahlen einliest und jeden dieser Parameter einzeln in einer Zeile ausgibt. Machen Sie sich zunächst mit dem Compiler, den Sie für die Sprache benötigen, vertraut – alle Compiler sind auf *marvin* verfügbar. Für Details konsultieren Sie die Abgaberichtlinien; dort werden auch benötigte Compiler-Switches erwähnt.

Sobald Sie den Code erfolgreich kompiliert haben, können Sie sich der Aufgabe widmen, die darin besteht, den Code so abzuändern, dass er die oben beschriebene Funktionalität implementiert.

Aufgabe 1.1 ..... 2 Punkte

Dateiname: `eins.c`

In dieser Aufgabe sollen Sie Ihr Programm in der Programmiersprache **C** implementieren. Folgender Code steht Ihnen als Basis zur Verfügung:

```c
#include<inttypes.h>
#include<stdio.h>

int main(int argc, char* argv[]) {
  puts("Got the following arguments:");
  for (int i = 1; i < argc; ++i) {
    const int val = (int)strtoimax(argv[i], NULL, 10);
    printf(" - %d\n", val);
  }
}
```

Aufgabe 1.2 ..... 2 Punkte

Dateiname: `Zwei.java`

Nun sollen Sie Ihr Programm in der Programmiersprache **Java** implementieren. Folgender Code steht Ihnen als Basis zur Verfügung:

```java
import java.lang.Integer;
import java.lang.String;
import java.lang.System;

public class Zwei {
    public static void main(final String[] args) {
        System.out.println("Got the following arguments:");
        for (final String arg: args) {
            final int val = Integer.parseInt(arg);
            System.out.println(" - " + Integer.toString(val));
        }
    }
}
```

Aufgabe 1.3 ..... 2 Punkte

Dateiname: `drei.adb`

Diesmal sollen Sie Ihr Programm in der Programmiersprache **Ada** implementieren. Folgender Code steht Ihnen als Basis zur Verfügung:

```
1  with Ada.Command_Line;
2  with Ada.Text_IO;
3  with Ada.Integer_Text_IO;
4
5  procedure Drei is
6     subtype Result is Natural range 0 .. 10;
7     type Numbers is array (Integer range <>) of Result;
8
9     function Numbers_From_Command_Line return Numbers is
10       use Ada.Command_Line;
11    begin
12       return Values : Numbers (1 .. Argument_Count) do
13          for I in 1 .. Argument_Count loop
14             Values (I) := Result'Value (Argument (I));
15          end loop;
16       end return;
17    end Numbers_From_Command_Line;
18
19    Points : constant Numbers := Numbers_From_Command_Line;
20 begin
21    Ada.Text_IO.Put_Line ("Got the following arguments:");
22    for I in Points'Range loop
23       Ada.Text_IO.Put (" - ");
24       Ada.Integer_Text_IO.Put (Points (I), Ada.Text_IO.Field'First);
25       Ada.Text_IO.New_Line;
26    end loop;
27 end Drei;
```

Aufgabe 1.4 ..... 3 Punkte

Dateiname: `vier.hs`

Schließlich sollen Sie Ihr Programm in der Programmiersprache **Haskell** implementieren. Folgender Code steht Ihnen als Basis zur Verfügung:

```
1  import System.Environment
2
3  process :: [Integer] -> String
4  process l = "Got the following arguments:\n" ++
5              concat (map (\x -> " - " ++ (show x) ++ "\n") l)
6
7  main = getArgs >>= putStr . process . (map read)
```

Hinweis: Ein- und Ausgabe in Haskell sind relativ komplex. Für diese Aufgabe ist es nicht notwendig, Zeit darin zu investieren, den Aufbau der `main`-Funktion zu verstehen. Benutzen Sie `process` als Ausgangspunkt für Ihre Modifikationen.