Dr. S. Lewandowski **Einführung in die Informatik I (autip)** WS 2005/06

Übungsblatt 04

Ausgabe: 16.11. Abgabeschluss: Mittwoch, 23.11., 9:45 Uhr, `eClaus.informatik.uni-stuttgart.de`

Abgabe erfolgt ausschließlich elektronisch über `eClaus.informatik.uni-stuttgart.de` – versuchen Sie nach Möglichkeit die Abgabe nicht in der letzten Minute zu machen!

Von jedem Aufgabenblatt werden maximal 20 Punkte auf den Schein angerechnet.

Es wird gebeten, bei den Freitext-Aufgaben, die Abgabe bitte NICHT als Word-Datei einzusenden. Bitte entweder per Copy-&-Paste übertragen oder als Plain-Text-Datei oder in den Formaten PDF bzw. PostScript.

Bei den Programm-Dateien fehlte überraschend häufig die Dateiendung. Bitte darauf achten, dass die Ada-95-Dateien die Dateiendung „.adb“ haben.

Ab diesem Übungsblatt 04 sind offiziell Abgaben in Gruppen bis zu 4 Personen zugelassen. Wählen Sie dazu beim ersten Bearbeiten einer Aufgabe im eClaus die Studierenden, mit denen Sie gemeinsam abgeben wollen, aus der Liste aller Teilnehmer aus. Die Bearbeitung ist von jedem der Abgabegruppe im eClaus einzusehen und kann von jedem verändert werden. Abgabegruppen gelten immer für 1 Übungsblatt. Sie müssen bei jedem neuen Übungsblatt die Mitglieder Ihrer Abgabegruppe erneut auswählen, auch wenn sich diese gegenüber dem letzten Übungsblatt nicht geändert haben.

Jede(r) einer Abgabegruppe muss alle Abgaben dieser Gruppe verstanden haben und vorrechnen können – ggf. werden der gesamten Gruppe die Punkte aberkannt.

Kopierte Abgaben werden ebenfalls ab diesem Aufgabenblatt strenger bewertet. Bei offensichtlich kopierten Abgaben werden die Punkte jeweils halbiert.

Ein paar Worte noch zu den Zusatzaufgaben: Diese richten sich insbesondere an Studierende, die schon gute Programmier(vor)kenntnisse haben. Sie dienen hier als Anreiz, eigene Grenzen auszuloten. Der Inhalt der Zusatzaufgaben ist für die Klausuren nicht relevant! Konzentrieren Sie sich also zunächst auf die Lösung der eigentlichen Aufgaben.

---

1. (2 Punkte, leicht) **Typ-Umwandlungen:** Die Typ-Umwandlung von `float` nach `integer` rundet mathematisch, d.h. ab „.5“ wird aufgerundet. In manchen Anwendungen möchte man jedoch einfach die Nachkommastellen ignorieren und mit einer `integer`-Zahl weiterrechnen. Schreiben Sie solch eine Funktion zur Typ-Umwandlung von `float` nach `integer`. Überlegen Sie sich ausreichend Testfälle und fügen Sie die entsprechenden Ausgaben im Hauptprogramm in der Form „`Die Zahl 4.567 lautet ohne Nachkommastellen 4`“ mit ein.

2. (2+2 Punkte) **Rolle Rückwärts**

   (a) (2 Punkte, leicht-mittel) Schreiben Sie eine rekursive Prozedur, die solange Zahlen einliest, bis die Zahl 0 eingeben wurde, und die eingegebenen Zahlen danach in umgekehrter Reihenfolge wieder ausgibt. Bei Eingabe der Zahlen 5 <Return>, 12 <Return>, 7 <Return>, 0 <Return> soll die Ausgabe 7 12 5 lauten. Zur Lösung dieser Aufgabe sind die bisher gelernten Sprachelemente von Ada 95 ausreichend!
   Fügen Sie Ihre Idee in den Programmkopf als Kommentarzeilen mit ein und kommentieren Sie Ihr Programm. Achten Sie auch auf sinnvolle Einrückungen.

   (b) (2 Punkte) Bei der einfachsten Lösung der Teilaufgabe (a) sind die eingebenen Zahlen nach der Ausgabe verloren. Wie müssten Datenstrukturen aussehen, um die Zahlen danach noch weiterverarbeiten zu können? Muss die Anzahl der einzugeben Zahlen dazu vorher bekannt sein? Skizzieren Sie eine mögliche Lösung. (Eine Variante werden wir bald in der Vorlesung kennenlernen, eine weitere später im Semester)

3. (4 Punkte) **Phythagoräische Zahlen-Tripel:** Der Satz von Phythagoras besagt, dass in einem rechtwinkligen Dreieck die Summe der Quadrate der Katheten gleich dem Quadrat der Hypothenuse ist. Im Allgemeinen sind in einem rechtwinkligen Dreieck nicht alle drei Kantenlängen ganzzahlig. Es gibt jedoch Beispiele, bei denen dies der Fall ist, z.B. $3^2 + 4^2 = 5^2$.

   (a) (4 Punkte, leicht) Schreiben Sie ein Programm, das eine Zahl n einliest und alle Zahlen-Tripel (a, b, c) mit $1 \leq a \leq b \leq c \leq n$ ausgibt. Für $n = 20$ soll die Ausgabe z.B. so aussehen:

```
Berechnung aller Phythagoraeischen Zahlen-Tripel (a,b,c) mit 1<=a<=b<=c<=20:
   1. Zahlen-Tripel: (3,4,5)
   2. Zahlen-Tripel: (5,12,13)
```

```
3. Zahlen-Tripel: (6,8,10)
4. Zahlen-Tripel: (8,15,17)
5. Zahlen-Tripel: (9,12,15)
6. Zahlen-Tripel: (12,16,20)
```

(b) **Zusatzaufgabe (6 Punkte, sehr schwer, nur für Tüftler)**: Der Aufwand bei Teil (a) ist bei der einfachen Lösung proportional zu n^3. Wir wollen dies hier nun wesentlich beschleunigen.

Für alle ungeraden n ist $(n, (n^2-1)/2, (n^2+1)/2)$ ein Phythagoräisches Zahlen-Tripel (d.h. auch, es gibt unendlich viele solcher Zahlen-Tripel, die nicht nur Vielfache voneinander sind), dabei beträgt die Differenz zwischen der zweiten und dritten Zahl stets genau 1.

Entwickeln Sie einen Ausdruck, der die Phythagoräischen Zahlen-Tripel mit Differenz 2 zwischen der zweiten und dritten Zahl beschreibt, verallgemeinern Sie dies (Hinweis: Die so gefundenen Zahlen sind nicht immer ganzzahlig, bei der Ausgabe müssen solche Tripel dann ausgelassen werden).

Schreiben Sie mit dem so entwickelten Ausdruck ein Programm zur Berechnung der gesuchten Zahlen-Tripel. Vergleichen Sie experimentell die Laufzeiten von der einfachen und Ihrer neuen Lösung, fügen Sie den Vergleich als Kommentarzeilen mit ein. Können Sie auch mathematisch abschätzen, wie groß der Aufwand zur Berechnung nun ist?

4. (3+4+3 Punkte) **Zahlen raten:** Ein einfaches Spiel ist, sich eine ganze Zahl (z.B. zwischen 0 und 100) auszudenken und den Gegenspieler diese Zahl erraten zu lassen. Als Hinweis verrät man dem Gegenspieler bei jedem Raten nur, ob die geratene Zahl mit der ausgedachten Zahl übereinstimmt, kleiner oder größer ist.

(a) (3 Punkte, mittel) Spielen Sie dieses Spiel zunächst untereinander und versuchen Sie Ihr Vorgehen dabei umgangssprachlich als Algorithmus zu formulieren (kein Ada-95-Programm). Begründen Sie, warum Ihr Vorgehen beim Raten effizient ist.

(b) (3+1 Punkte, mittel) Schreiben Sie ein Programm, das einen Spieler zunächst eine Zahl einlesen lässt. Der andere Spieler soll nun versuchen diese zu erraten. Das Programm gibt Hinweise, ob die geratene Zahl stimmt, kleiner oder größer ist. Ist die geratene Zahl außerhalb des Bereichs, der aufgrund der bisherigen Tipps schon bekannt ist, so soll darauf hingewiesen werden. (3 Punkte)

(+1 Punkt) Statt die Zahl von dem einen Spieler eingeben zu lassen, können Sie sich von folgendem Programm inspirieren lassen, wie man mit Ada 95 Zufallszahlen erzeugen kann.

```
with Ada.Text_IO, Ada.Numerics.Float_Random;
use Ada.Text_IO, Ada.Numerics.Float_Random;

procedure RandDemo is
  G : generator;
  Zufall : float;
begin
  Reset (G);
  for i in 1..20 loop
    Zufall := Random (G);
    if Zufall < 0.5 then Put_Line("Kopf"); else Put_Line("Zahl"); end if;
  end loop;
end RandDemo;
```

Details finden sich im Abschnitt A.5.2 des Reference Manuals, grob gesagt ist folgendes zu tun: Ein Objekt vom Typ `Generator` deklarieren, dieses *einmal* als Parameter von `Reset` verwenden, anschließend beliebig oft als Parameter von `Random`, das Ergebnis letzterer Funktion ist dann ein „zufälliger“ Wert im Bereich zwischen 0 und 1. (Da die direkte Möglichkeit ganzzahlige Zufallszahlen in Ada 95 zu erzeugen etwas komplizierter ist, begnügen wir uns hier mit dieser etwas einfacheren Variante – schreiben Sie sich ggf. eine `function Random_Int(n:natural) return natural` selbst, die eine Zufallszahl zwischen 0 und `n-1` erzeugt.)

(c) (3 Punkte, mittel) Erweitern Sie Ihr Programm aus Teil (b) nun so, dass der Computer die Zahl errät, die Sie sich ausdenken (diese soll dann von Ihnen im Programm nicht mehr eingegeben werden). Der Computer soll dabei erkennen, wenn Sie ihn durch widersprüchliche Angaben in die Irre führen wollen.

Zu Beginn soll das Programm fragen, ob der Computer oder wie in Teil (b) der Benutzer raten soll.
