Programmierkurs I (Inf., W-Info.), Übungsblatt 10

Claus/Weicker, Wintersemester 03/04 Abgabetermin: 15.01.2004, 23:59 Uhr

Bitte beachten Sie die Hinweise von Aufgabenblatt 8 sowie die Angaben in den jeweiligen Spezifikationsdateien ∗.ads.

Aufgabe 1: π berechnen (mittel) — 6 Punkte

Wie Sie wissen, bezeichnet die Zahl π die Fläche des Einheitskreises, d.h. des Kreises mit Mittelpunkt $(0, 0)$ und Radius 1. Umgekehrt heißt das, dass man π berechnen kann, indem man die Fläche des Einheitskreises misst. Dazu kann man sich einer wahrscheinlichkeitstheoretischen Methode (einer sogenannten Monte-Carlo-Methode) bedienen. Diese geht wie folgt:

> Wähle n-mal einen Punkt (x, y) zufällig gleichverteilt aus dem Quadrat $[-1, 1] \times [-1, 1]$ aus. Falls (x, y) innerhalb des Einheitskreises liegt, nennen wir das einen Treffer. Wenn n gegen unendlich tendiert, wird das Verhältnis der Treffer zur Anzahl der ausgewählten Punkte sich dem Verhältnis der Fläche des Einheitskreises zu der des Quadrats $[-1, 1] \times [-1, 1]$, d.h. $\pi/4$ annähern.

Implementieren Sie die in der Datei `piberechnung.ads` vorgegebene Funktion

```
function Pi_Naeherung(N: Natural) return Float;
```

und geben Sie die zugehörige Datei `piberechnung.adb` ab. `N` sei dabei die Anzahl der Punkte, die zufällig gewählt werden.
Hinweis 1: Der Abstand zwischen zwei Koordinaten (x_1, y_1) und (x_2, y_2) ist die übliche euklidische Distanz

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Hinweis 2: Für (Pseudo-)Zufallszahlen gibt es das Paket `Ada.Numerics.Float_Random`. Wenn Sie dieses benutzen und eine Variable `seed:Generator` deklarieren, liefert der Ausdruck `random(seed)` eine zufällige reelle Zahl im Bereich von 0 bis 1.

Aufgabe 2: Stammbaum (mittel) — 14(+4) Punkte

Die Stammbauminformationen mehrerer Familien sollen von einer Software verwaltet werden, die verschiedene Anfragen an die Daten erlaubt. Jeder Person ist ein eindeutiger Schlüssel, d.h. eine natürliche Zahl, zugeordnet, mit der die Person identifiziert wird. Auf der Vorlesungsseite ist eine Datei `stammbaum.txt` erhältlich, die Daten in folgendem Format enthält: Jede Zeile entspricht einem Datensatz, z.B.

```
247#Napoleon#249#330
```

Vier Felder sind durch das Sonderzeichen `#` voneinander abgetrennt. Das erste Feld enthält den Schlüssel der Person, das zweite Feld den Namen, das dritte Feld den Schlüssel der Mutter und das vierte Feld den Schlüssel des Vaters.
Die Datei `familie.ads` enthält die Spezifikation der Schnittstelle zur Verwaltungssoftware. Implementieren Sie diese in der Datei `familie.adb`.

- Entwerfen Sie einen Datentypen als Record, um die Daten einer Person zu speichern. Legen Sie in der Datei `familie.adb` eine globale Variable im Paket an, die ein Array (Feld) mit 48 Einträgen des Datentyps enthält. Der Inhalt dieser Variablen wird von den Prozeduren und Funktionen modifiziert oder ausgelesen. Den Namen sollten Sie in einem String ablegen, für den Sie eine maximale Länge von 40 Zeichen vorsehen können. Füllen Sie gegebenenfalls den String mit Leerzeichen auf.

- Datenimport (4 Punkte): Schreiben Sie eine Funktion

  ```
  function Importiere_Daten (Dateiname: String) return Boolean;
  ```

um die Daten aus einer Datei mit dem angegebenen Dateinamen einzulesen. Falls dies erfolgreich war, wird der Wert `true`, sonst der Wert `false` zurückgegeben (z.B. falls die Datei mehr als 48 Einträge enthält). Sie können ferner davon ausgehen, dass eine Zeile maximal 50 Zeichen enthält.

- Namen und Schlüssel (3 Punkte): Schreiben Sie Prozeduren

```
procedure Liefere_Name (Schluessel: in Natural;
                        Name: out String;
                        Erfolg: out Boolean);
procedure Liefere_Schluessel (Name: in String;
                              Schluessel: out Natural;
                              Erfolg: out Boolean);
```

  um zum eindeutigen Schlüssel den Namen zu erhalten (sowie für die Gegenrichtung). Der Rückgabeparameter enthält `true` genau dann, wenn der entsprechende Datensatz gefunden werden konnte.

- Direkte Verwandte (3 Punkte): Schreiben Sie Funktionen, um die direkten Verwandten zu ermitteln.

```
function Mutter_Von (Schluessel: Natural) return Natural;
function Vater_Von (Schluessel: Natural) return Natural;
function Anzahl_Kinder (Schluessel: Natural) return Natural;
function Kind_Von (Schluessel: Natural;
                   Nummer: Natural) return Natural;
```

  Falls der entsprechende Verwandte nicht in den Daten enthalten ist, wird der Wert 0 zurückgegeben. Die Zahl `Nummer` in der letzten Funktion kann Werte von 1 bis Anzahl der Kinder annehmen – die Schlüssel der Kinder werden entsprechend aufsteigend ausgegeben. Falls ein fehlerhafter Wert für `Nummer` eingegeben wird, wird ebenfalls 0 zurückgegeben.

- Gemeinsame Vorfahren (4 Punkte): Schreiben Sie eine Prozedur, die berechnet, ob zwei Personen einen gemeinsamen Vorfahren besitzen.

```
function Gemeinsamer_Ahn (Schluessel1, Schluessel2: Natural) return Boolean;
```

  Die Funktion gibt genau dann `true` zurück, wenn ein gemeinsamer Vorfahre existiert. So ist im Datensatz beispielsweise Faruk ein gemeinsamer Ahn von Dagobert und Laila.
  Hinweis: Sie können diese Aufgabe lösen, indem Sie rekursiv die Eltern einer Person markieren und dann rekursiv für die andere Person kontrollieren, ob Sie einen markierten Datensatz finden.

- **Zusatzaufgabe** Gemeinsame Nachkommen (4 Punkte): Analog können Sie die Funktion

```
function Gemeinsame_Nachkommen (Schluessel1, Schluessel2: Natural)
                                return Boolean;
```

  implementieren, um zu prüfen, ob es wenigstens einen gemeinsamen Nachkommen gibt. In den Daten wäre etwa Faralda ein gemeinsamer Nachkomme von Adele und Vanadis.

Hinweise

- Zur Abgabe wird das eClaus-System verwendet:
  `http://eclaus.informatik.uni-stuttgart.de`

- Die Abgabe zu jeder Teilaufgabe besteht aus einem kompilierbaren Ada-Quelldatei. In jeder Aufgabe wird ein Dateiname vorgegeben. Verwenden Sie diesen bitte für das Hauptprogramm. Auch sind in der Aufgabe Angaben zu Ein- und Ausgabetexten sowie zur Formatierung der Ausgabe enthalten. Bitte folgen Sie diesen Angaben so genau wie möglich.

- Beachten Sie beim Programmieren bitte die folgenden Hinweise („kleine Programmierrichtlinie“).
  - Halten Sie einzelne Bestandteile überschaubar, z.B. indem Sie nur eine Anweisung pro Zeile schreiben, sowie pro Zeile höchstens 80 Zeichen, höchstens 40 Zeilen pro Prozedur/Funktion, nicht mehr als 800 Zeilen pro Datei und höchstens 5 Parameter bei Prozeduren/Funktionen benutzen.
  - Bezeichner sollen selbsterklärend und maximal 15 Zeichen lang sein. Bezeichner enthalten nur Buchstaben, den Bindestrich oder den Unterstrich.
  - Durch Einrückung um 2 Zeichen soll die logische Gliederung eines Programms verdeutlicht werden. Beispielsweise wird der Anweisungsteil einer IF-Verzweigung eingerückt, während „`end if;`“ nicht mehr eingerückt wird. Auch auf der nächsten Zeile fortgesetzte Zeilen werden um 2 Zeichen eingerückt.
  - Zu Beginn des Programms muss in Kommentaren das Konzept und die Lösungsidee des Programms ausführlich erläutert werden.
  - Auch im Programmtext sind Kommentare einzufügen, um den Code zu erläutern.
- Beachten Sie, dass jede Abgabe individuell vom jeweiligen Studierenden erstellt werden muss. Werden von den Tutoren Plagiate erkannt, d.h. exakte oder leicht modifizierte Kopien, werden für die Aufgabe keine Punkte vergeben. Falls Sie die Lösung der Aufgaben vor der Abgabe in Gruppen besprechen, achten Sie darauf, dort nur das generelle Konzept abzuklären und die Programmierung jedem selbst zu überlassen.
- Bei Fragen wenden Sie sich bitte an Ihren Tutor oder an die Übungsleitung:
  `Karsten.Weicker@fmi.uni-stuttgart.de` oder Tel. 7816-337
- Weitere Veranstaltungshinweise einschließlich der Übungsblätter finden Sie unter:
  `http://www.informatik.uni-stuttgart.de/ifi/fk/lehre/ws03-04/ada95/`