Programmierkurs I (Inf., W-Info.), Übungsblatt 8

Claus/Weicker, Wintersemester 03/04 Abgabetermin: 18.12.2003, 23:59 Uhr

Wichtige Hinweise: Ab diesem Übungsblatt werden keine kompletten Programme mehr als Abgaben erwartet sondern einzelne Prozeduren und Funktionen. Dies hat den großen Vorteil, dass bei der Überprüfung der Korrektheit nicht mehr die Ausgaben des Programms überprüft werden, sondern die Ausgabeparameter der Prozeduren bzw. Rückgabewerte der Funktionen direkt mit den erwarteten Werten verglichen werden.
Zunächst werden für alle Aufgaben verschiedene Dateien vorgegeben, die die Bearbeitung der Aufgaben vereinfachen. So ist immer eine Spezifikationsdatei mit der Endung `.ads` gegeben, welche die benötigten Datentypen und Prozeduren/Funktionen mit ihren Ein- und Ausgaben definiert. In einer zugehörigen Implementationsdatei `.adb` wird der Ada-Programmcode für die Prozeduren und Funktionen ausgeführt. Zunächst sollen Sie nur an den angegebenen Stellen in den Implementationsdateien Änderungen vornehmen.
In diesem Aufgabenblatt sind auch beispielhaft Testdateien vorgegeben, die Ihnen eine Idee vermitteln, wie Sie Ihre Implementation der gewünschten Funktionen und Prozeduren überprüfen können.
Die Abgabe besteht ausschließlich aus den Dateien, die für Ihre veränderte adb-Datei benötigt werden. D.h. im Regelfall ist dies ausschließlich die adb-Datei. Da Sie keine Änderungen in der Spezifikation der ads-Datei vornehmen sollen, braucht diese nicht mit abgegeben zu werden (außer dies wird auf einem der folgenden Blätter ausdrücklich verlangt). **Bitte verändern Sie weder den Dateinamen der abzugebenden Datei noch den Namen des darin implementierten Pakets.** Auch ist Ihr Testtreiber nicht Teil der der Abgabe. Die automatische Überprüfung Ihrer Programme benutzt andere Testtreiber, die lediglich die von Ihnen implementierten Prozeduren und Funktionen einbinden.

Aufgabe 1: Anagramme II (mittel) — 6 Punkte

Eine Zeichenkette t nennt man ein Anagramm einer Zeichenkette s, wenn man t erhalten kann, indem man die Zeichen von s in einer neuen Reihenfolge anordnet. Zum Beispiel ist `internet` eines von `renitent` und `nikolaustag` eines von `glasauktion`.
Man kann diesen Sachverhalt auch mathematisch präzise ausdrücken: Haben s und t die gleiche Länge n und sind von der Form $s = s_1 \dots s_n$ und $t = t_1 \dots t_n$, so sind s und t genau dann Anagramme voneinander, wenn es eine bijektive Funktion $f : \{1, \dots, n\} \to \{1, \dots, n\}$ (d.h. eine Permutation) gibt, so dass $s_i = t_{f(i)}$ gilt für alle $1 \leq i \leq n$.
Schreiben Sie eine Prozedur, die überprüft ob zwei eingegebene Zeichenketten Anagramme sind und geben Sie eine gültige Permutation aus, wie die Buchstaben des ersten Wortes angeordnet werden können, um das zweite Wort zu formen. Modifizieren Sie hierzu die Datei `anagramm2.adb` von der Veranstaltungsseite. Dort sind sowohl der Datentyp

```
type Permutation is array (Integer range <>) of Integer;
```

als auch die Prozedur

```
procedure Teste_Anagramm (S: in String; T: in String;
                          Ist_Anagramm: out Boolean;
                          Anordnung: out Permutation);
```

definiert. Ihre Aufgabe ist es, die Prozedur so zu implementieren, dass bei einem Aufruf

```
Teste_Anagramm ("internet", "renitent", Erfolgreich, Perm);
```

(mit der boolschen Variable `Erfolgreich` und der Variable `Perm` vom Datentyp `Permutation`) die Permutation die Werte (5, 4, 2, 1, 3, 7, 6, 8), (5, 4, 6, 1, 3, 7, 2, 8), (5, 4, 2, 1, 8, 7, 6, 3) oder (5, 4, 6, 1, 8, 7, 2, 3) enthält. Verzichten Sie bei Ihrer Implementation auf die Verwendung des Datentyps `Unbounded_String`.
Für Tests Ihrer Implementation können Sie die Datei `test_ana.adb` benutzen und für Ihre Zwecke modifizieren. Diese Datei ist jedoch nicht Teil Ihrer Abgabe.

Aufgabe 2: Game of Life (mittel) 14 Punkte

Bei dem von dem Mathematiker Conway erfundenen 'Game of Life' handelt es sich nicht um ein Spiel im klassischen Sinne, sondern um eine Simulation. Die Simulation spielt auf einer Matrix von $m \times n$ 'Zellen'; jede Zelle ist entweder 'bewohnt' oder 'unbewohnt'. Ausgehend von einem Anfangszustand ändert sich der Zustand der Zellen von Generation zu Generation, und zwar nach den folgenden Regeln:

- Ist eine Zelle in einer Generation bewohnt und hat nur eine oder gar keine bewohnte Nachbarzelle, so ist sie in der nächsten Generation unbewohnt (man sagt: sie stirbt an Einsamkeit).
- Ist eine Zelle in einer Generation bewohnt und hat mehr als drei bewohnte Nachbarzellen, so ist sie in der nächsten Generation ebenfalls unbewohnt (sie stirbt an Überbevölkerung).
- Ist eine Zelle in einer Generation unbewohnt und hat genau drei bewohnte Nachbarn, so ist sie in der nächsten Generation bewohnt.
- In allen anderen Fällen ändert sich der Zustand einer Zelle von einer Generation zur anderen nicht.

Als Nachbarn einer Zelle an Position (x, y) zählen genau die acht Zellen, deren Positionen sich in keiner Dimension um mehr als 1 von x bzw. y unterscheiden (außer der Zelle selbst). In dieser Aufgabe gehen wir von einer 100×100-Matrix aus und nehmen ferner an, dass die Ränder der Zellen zusammengeklebt sind, d.h. die Zelle $(1, 5)$ hat die Nachbarn $(1, 4)$, $(1, 6)$, $(2, 4)$, $(2, 5)$, $(2, 6)$, $(100, 4)$, $(100, 5)$ und $(100, 6)$. Die Zelle $(100, 100)$ hat die Nachbarn $(99, 99)$, $(99, 100)$, $(99, 1)$, $(100, 99)$, $(100, 1)$, $(1, 99)$, $(1, 100)$ und $(1, 1)$.
Modifizieren Sie die Datei `gameoflife.adb` und implementieren Sie die folgenden Prozeduren

- `Brett_Speichern`: Der Inhalte des Spielbretts ist so in einer Datei abzuspeichern, dass die vorgegebene Prozedur `Brett_Laden` die Datei wieder einlesen kann. (3 Punkte)
- `Naechste_Iteration`: Für alle Zellen werden die neuen Werte für die Belegung berechnet. Achten Sie darauf, dass die neu errechneten Werte zwischengespeichert werden müssen, da Sie die aktuellen Werte noch für die Berechnung anderer Zellen benötigen. (9 Punkte)
- `Brett_Drucken`: Schreiben Sie eine Prozedur, die den Inhalt des Bretts in einem beliebigen Format ausgibt. Sie werden diese Prozedur evtl. für den Test Ihrer Programme benötigen. Die Funktionalität dieser Prozedur wird nicht automatisch überprüft, so dass Sie keine Vorgaben beachten müssen.

Führen Sie Tests mit Ihrer Implementation durch. Benutzen Sie hierfür beispielsweise das Programm `test_gol.adb`, das Sie für weitere Tests beliebig verändern können. Auf der Veranstaltungsseite ist auch eine Datei `Ein_Brett.txt` erhältlich, die ein abgespeichertes Spielbrett enthält. `test_gol.adb` ist nicht Teil Ihrer Abgabe.

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/`