

# Programmierübungen

Wintersemester 2007/2008

## 9. Übungsblatt

11. Januar 2008

Abgabe bis Freitag, 18. Januar 23:59 Uhr

Die Abgabe Ihrer Bearbeitung können Sie im eClaus-System durchführen. Erarbeiten Sie Lösungsideen zu den Aufgaben möglichst in Kleingruppen. Es wird jedoch von Ihnen erwartet, dass jeder Teilnehmer eine eigene Lösung abgibt. Sollten kopierte Quelltexte abgegeben werden, so werden grundsätzlich alle Kopien mit 0 Punkten bewertet. In den Vortragsfolien der Programmierübungen oder im Skript zur Einführung in die Informatik abgedruckte Quelltexte können verwendet werden, müssen aber der Programmierrichtlinie entsprechend formatiert und kommentiert werden.

Beachten Sie die Programmierrichtlinie und kommentieren Sie Ihren Quelltext! Dokumentieren Sie unbedingt Ihre Lösungsidee in den Quelltext-Kommentaren.

<http://www.iste.uni-stuttgart.de/ps/Lehre/WS0708/inf-prokurs>

*Bitte beachten Sie, dass zum Scheinerwerb unter anderem auf den letzten 4 Aufgabenblättern 50% der Punkte erreicht werden müssen. Das letzte Aufgabenblatt wird Nummer 12 sein.*

### Aufgabe 9.1: Median-Berechnung

(7 Punkte)

Sei  $(x_1, x_2, \dots, x_n)$  mit  $x_i \leq x_{i+1}$  für alle  $1 \leq i < n$  eine sortierte Folge von  $n$  Zahlen. Falls  $n$  ungerade ist, so ist der Obermedian dieser Folge definiert als das mittlere Element der Folge. Ist  $n$  gerade, so ist der Obermedian das rechte Element des in der Mitte stehenden Paares. Der Obermedian ist also stets das Element  $x_{\lceil \frac{n+1}{2} \rceil}$ .

Erstellen Sie ein Paket `Median`, das den abstrakten Datentyp `Number_Sequence` enthält. Verwenden Sie die folgende Vorlage für die Spezifikation des Pakets, ergänzen Sie den privaten Teil der Spezifikation, übernehmen Sie den öffentlichen Teil unverändert:

```
package Median is

  type Number_Sequence is private;

  procedure Make
    (Sequence : in out Number_Sequence);

  procedure Add_Number
    (Sequence : in out Number_Sequence;
     Number   : in Integer);

  No_Numbers : exception;

  function Result
    (Sequence : in Number_Sequence)
    return Integer;

  procedure Reset
    (Sequence : in out Number_Sequence);
```

**private**

-- ...

**end** Median;

Ein Hauptprogramm zur Medianberechnung verwendet das Paket Median auf die folgende Weise:

1. Ein Aufruf von Make erzeugt ein Objekt des Typs Number\_Sequence. Dieses Objekt repräsentiert eine Zahlenfolge wie oben beschrieben.
2. Durch wiederholten Aufruf der Prozedur Add\_Number werden dem Number\_Sequence-Objekt Zahlen hinzugefügt.
3. Ein Aufruf der Funktion Result berechnet den Obermedian des Number\_Sequence-Objekts. Wurden dem Number\_Sequence-Objekt keine Zahlen hinzugefügt, so erhebt die Funktion Result die Ausnahme No\_Numbers
4. Durch einen Aufruf von Reset können alle Zahlen eines Number\_Sequence-Objekts gelöscht werden.

Das Paket *muss* beliebig lange Folgen von Zahlen verarbeiten können. Es soll möglich sein gleichzeitig mehrere Number\_Sequence-Objekte unabhängig voneinander mit Zahlen zu füllen und deren Obermedian zu bestimmen.

## **Aufgabe 9.2: Baum-Gleichheit**

**(4+6 Punkte)**

In dieser Aufgabe soll eine Funktion implementiert werden, die zwei Bäume auf Gleichheit überprüft. Zu diesem Zweck wird angenommen, dass jeder Knoten eines Baums Daten in Form eines Integer-Werts speichert. Ein Knoten soll eine Anzahl Kindknoten besitzen können, die in einer bestimmten Reihenfolge angeordnet sind.

Gegeben seien zwei Bäume mit Wurzelknoten  $A$  und  $B$ .  $A$  habe die Kindknoten  $a_1, a_2, \dots, a_m$ , und  $B$  habe die Kindknoten  $b_1, b_2, \dots, b_n$ . Die beiden Bäume sind gleich, falls ...

- die Wurzelknoten  $A$  und  $B$  mit den gleichen Daten beschriftet sind,
- die beiden Wurzelknoten gleich viele Kindknoten besitzen, d. h.  $n = m$  gilt, und
- die entsprechenden Teilbäume definiert durch die Kinder gleich sind, d. h. der Teilbaum mit Wurzel  $a_i$  gleich dem Teilbaum mit Wurzel  $b_i$  ist für alle  $1 \leq i \leq n$ .

Auf der Webseite werden die Pakete Binary\_Tree\_Equivalence und Tree\_Equivalence zum Download angeboten. In diesen Paketen sind jeweils nur die Datentypen und die Funktion Equals spezifiziert. In Binary\_Tree\_Equivalence werden jedoch nur Binärbäume betrachtet und in Tree\_Equivalence allgemeine Bäume mit beliebiger Anzahl Kinder (modelliert durch eine Liste von Kindern an jedem Knoten). Erstellen Sie die Implementierung der Equals-Funktionen in *beiden* Paketen. Beginnen Sie mit der Variante für Binärbäume (4 Punkte) und erweitern Sie Ihr Verfahren dann zur Behandlung von allgemeinen Bäumen (6 Punkte).

Hinweise:

- Wählen Sie eine Durchlauf-Strategie (z. B. Preorder) und lassen Sie Ihr Programm durch beide zu vergleichenden Bäume parallel durchlaufen. Lassen Sie das Programm abbrechen, sobald ein Unterschied zwischen den Bäumen gefunden wird.
- Fügen Sie den Paketen weitere Unterprogramme hinzu, um Ihre Implementierung zu testen

### Aufgabe 9.3: Zyklenerkennung

(3 Punkte)

Betrachten Sie die Pakete `Acyclic_US_Lists` und `Cyclic_US_Lists`. Diese stellen beide einen abstrakten Datentyp `List` zur Speicherung von `Unbounded_Strings` zur Verfügung. Sie bieten die gleichen Unterprogramme an. In der Datei `us_lists.ads` wird eines der beiden Pakete umbenannt und ist dadurch zusätzlich unter dem Namen `US_Lists` bekannt.

Implementieren Sie die Funktion `Is_Acyclic`, deren Spezifikation durch folgende Datei `is_acyclic.ads` angegeben ist:

```
with US_Lists;
```

```
function Is_Acyclic
  (List : in US_Lists.List)
  return Boolean;
```

Die Funktion soll durch Verwendung der Listen-Operationen erkennen können, ob eine zyklische oder eine azyklische Liste übergeben wurde. Die Funktion soll `True` zurückgeben, falls die leere Liste übergeben wird. Gehen Sie davon aus, dass der Name `US_Lists` wahlweise das Paket `Acyclic_US_Lists` oder das Paket `Cyclic_US_Lists` umbenennen könnte. Ändern Sie `us_lists.ads` um Ihre Implementierung zu testen.