Programmierübungen

Wintersemester 2006/2007

10. Übungsblatt
12. Januar 2007

Abgabe bis Samstag, 20. 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/WS0607/inf-prokurs`

Am Montag 15.1.2007, 8:00 Uhr in V38.01 wird das Aufgabenblatt kurz vorgestellt und Sie haben Gelegenheit Fragen zu den Aufgaben zu stellen.

Bitte beachten Sie, dass zum Scheinerwerb unter anderem auf den letzten 4 Aufgabenblättern 50 % der Punkte erreicht werden müssen. Das letzte Aufgabenblatt wird Nummer 13 sein.

Aufgabe 10.1: Median-Berechnung (7 Punkte)

Sei $(x_1, x_2, \ldots, x_n)$ 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 Paars. Der Obermedian ist also stets das Element $x_{\lceil \frac{n+1}{2} \rceil}$.

Erstellen Sie ein Paket Median als abstraktes Datenobjekt mit folgender Spezifikation:

```
package Median is

   procedure Reset;

   procedure Add_Number
     (Number : in Integer);

   No_Numbers : exception;

   function Result
     return Integer;

end Median;
```

Durch wiederholten Aufruf der Prozedur Add_Number sollen ganze Zahlen an die anfangs leere Folge angefügt werden. Durch einen Aufruf der Funktion Result wird der Obermedian berechnet und zurück gegeben. Wird Result zu einem Zeitpunkt aufgerufen, zu dem keine Zahlen mit Hilfe von Add_Number eingegeben wurden, so soll Result die Ausnahme No_Numbers erheben. Durch einen Aufruf der Prozedur Reset soll das

Paket die (durch Aufrufe von Add_Number) bereits eingefügten Zahlen vergessen und erneut mit der leeren Folge starten.

Das Paket muss beliebig lange Folgen von Zahlen verarbeiten können.

Aufgabe 10.2: 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 $a_1, a_2, \ldots, a_m$, und B habe die Kindknoten $b_1, b_2, \ldots, b_n$. 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_Eqivalence 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 10.3: 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.