Dr. S. Lewandowski **Einführung in die Informatik I (autip)** WS 2006/07

Übungsblatt 11

Ausgabe: 24.01. Abgabeschluss: Mittw., 31.01., 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.

Allgemeine Hinweise für dieses Aufgabenblatt: Eine Datenstruktur zusammen mit den darauf definierten Operationen und ihren Eigenschaften nennt man in der Informatik einen **Abstrakten Datentyp** (ADT). Solche Datenstrukturen mit den zugehörigen Operationen bilden oft Einheiten, die in vielen Programmen wiederverwendet werden können. Ada bietet zum Einbinden von solchen Einheiten das Konzept der *Packages* an – wir haben diese schon lange verwendet (z.B. `Ada.Text_IO`, etc.). Geben sie die Datenstruktur und die Operationen in einem entsprechenden Paket ab. Das jeweilige Testprogramm soll dann Ihr entsprechendes Paket über `with` und `use` einbinden und benutzen.

Betrachten Sie als Beispiel das Paket mit den Grundfunktionen der Listen in Aufgabe 2 (dies steht auf der Vorlesungsseite zum Download bereit).

---

1. (1+2+2(+3)+5+3+2 Punkte, mittel) **Rechnen mit Brüchen:** Gleitkommazahlen bringen stets automatisch gewisse Rundungsfehler mit sich - so lassen sich z.B. selbst einfache Zahlen wie 0.2 oder 23.1 mit dem Datentyp `float` nicht exakt im Rechner darstellen. Wir wollen hier nun Abhilfe schaffen.

   Wir definieren uns einen Typ für Rationale Zahlen (Brüche), also ein Paar (Zähler,Nenner), wobei der Nenner stets positiv ist. Die Darstellung ist nicht eindeutig, da wir den Bruch 2/3 z.B. auch als 4/6 schreiben können, ohne den Wert dabei zu verändern. Um diese Mehrdeutigkeiten zu vermeiden, werden wir die Brüche stets soweit möglich kürzen. Den dazu benötigten größten gemeinsamen Teiler (ggT) von Zähler und Nenner können wir mit Euklids Algorithmus effizient berechnen. Die Zahl 0 soll durch 0/1 dargestellt sein.

   Haben wir die Grundrechenarten Plus, Minus, Mal und Geteilt implementiert, können wir ein paar Beispielprogramme schreiben.

   Nun ans Werk ... und vergessen Sie nicht, Ihre Programme zu kommentieren.

   - (1 Punkt) Definieren Sie einen Datentyp für Brüche, benutzen Sie dafür einen Verbund (Record).
   - (2 Punkte) Schreiben Sie Ein- und Ausgaberoutinen (Put und Get) für Ihren neu definierten Datentyp (Sie können Zähler und Nenner einzeln erfragen). Achten Sie darauf, dass Sie den Bruch nach Eingabe kürzen – nach Eingabe des Bruches 4/6, soll die Ausgabe der eingelesenen Zahl also 2/3 sein.

     (2 Punkte) Der Euklidische Algorithmus basiert auf den zwei Eigenschaften
     - ggT(a,b) = ggT(b,a mod b) für b größer 0 und
     - ggT(a,0) = a

     Achten Sie (bei negativen Zahlen) bei der ggT-Berechnung auf die Vorzeichen.

     Zusatzaufgabe (fürs Ego der Freaks): Zeigen Sie, dass der Aufwand zur Berechnung des ggT mit Euklids Algorithmus nur $O(\log(b))$ Schritte beträgt.
   - (5*1 Punkte) Implementieren Sie die 4 Grundrechenarten und die Negation.

- (3 Punkte) Berechnen Sie die Harmonische Funktion $h(n) := \sum_{i=1}^{n} 1/i$
  – **Zusatzaufgabe (fürs Ego der Freaks):** Hier kommt es schon bei relativ kleinem n zu constraint errors, weil die Nenner recht schnell recht groß werden. Verbessern Sie Ihre Algorithmen dahingehend, indem die Produkte soweit möglich schon vorab gekürzt werden, bevor die Multiplikation zu große Zahlen generiert. Bis zu welchem n können Sie nun die Harmonische Funktion berechnen?
- (2 Punkte) Demonstrieren Sie die Korrektheit der anderen Grundrechenarten durch einige Testfälle im Hauptprogramm.

2. (5*2 Punkte, mittel–schwer) **Listen:** Implementieren Sie die fehlenden Prozeduren und Funktionen aus der Vorlesung, Sie können hier als `Element_Typ` den Typ `Integer` verwenden.

   - `procedure add_to_end(list : in out Liste; elem : Element_Typ)` – fügt `elem` am Ende der Liste ein
   - `procedure add_sorted(list : in out Liste; elem : Element_Typ)` – fügt in eine vorher sortierte Liste `elem` so ein, dass danach die Liste wieder sortiert ist
   - `procedure delete(list:in out Liste; elem:Element_Typ; deleted:out Boolean)` – löscht das erste Vorkommen von `elem` in `list`, der Ausgangsparameter `deleted` wird `false` gesetzt, wenn `elem` nicht in `list` vorkam, sonst auf `true`
   - `procedure delete_all(list: in out Liste; elem : Element_Typ; deleted : out Natural)` – löscht alle Vorkommen von `elem` in `list` und zählt in `deleted`, wieviel Elemente gelöscht wurden
   - `function copy_List(list : Liste) return Liste` – kopiert die Liste

   Demonstrieren Sie die Korrektheit Ihrer Implementierungen mit entsprechenden Testfällen, die Sie in Ihr Hauptprogramm einbauen. Achten Sie auf ausreichend Kommentare und Beschreibung der Algorithmen in Ihrem Ada-95-Code.

   Beachten Sie auch die Hinweise zu diesen Prozeduren und Funktionen im Skript.

   Als Beispiel für Packages liegen auf der Vorlesungsseite die Dateien für das Package `listenblatt11.ads`, `listenblatt11.adb` sowie das Hauptprogramm `testblatt11.adb` bereit. Zum Übersetzen die Datei `testblatt11.adb` in AdaGide laden und übersetzen lassen (das Package wird dann automatisch eingebunden).
